

USER GUIDE

Essential Studio for EJ2 React

Version - v25.1.35 | Release Date - March 15, 2024

PivotTable	31
Getting started	31
Dependencies.....	31
Setup for Local Development.....	31
Adding Syncfusion packages	32
Adding CSS reference.....	33
Browser compatibility	33
Adding pivot table component	33
Adding fields to row, column, value and filter axes.....	36
Applying formatting to a value field	37
Module Injection.....	38
Enable Field List.....	39
Enable Grouping Bar	40
Exploring Filter Axis.....	41
Calculated Field.....	42
Run the application	44
Creating a Next.js Application Using Syncfusion React Components	45
What is Next.js?	45
Prerequisites	45
Create a Next.js application	45
Install Syncfusion React packages.....	46
Import Syncfusion CSS styles	46
Add Syncfusion React component	47
Run the application	48
Data binding in React Pivotview component.....	49
JSON	49
CSV	53
Remote Data Binding	57
Mapping	62
Values in row axis.....	65
Values at different positions	66
Show 'no data' items.....	67
Show value headers always	68
Customize empty value cells.....	69
Event	70

See Also	74
Connecting to data source	74
MySQL in EJ2 React Pivotview Component	74
Microsoft SQL Server in EJ2 React Pivotview Component	80
PostgreSQL in EJ2 React Pivotview Component	84
Oracle in EJ2 React Pivotview Component	89
MongoDB in EJ2 React Pivotview Component	95
Elasticsearch in EJ2 React Pivotview Component	100
Snowflake in EJ2 React Pivotview Component	105
Olap in React Pivot Table component	111
Getting started	111
Data Binding	145
OLAP Cube: Elements	155
Server side pivot engine in React Pivotview component	162
Quick steps to render the Pivot Table by using the server-side Pivot Engine	162
Available configurations in Server-side application	164
Pivot chart in React Pivotview component	183
Data Binding	185
Chart Types	185
Accumulation Charts	187
Field List	199
Grouping Bar	201
Single Axis	204
Multiple axis	205
Series customization	211
Axis customization	214
Legend customization	217
User Interaction	219
Export	225
Print	227
Drill down in React Pivotview component	228
Drill down and drill up	228
Drill position	229
Expand all	229
Expand all headers for specific fields	230

Expand all except specific member(s).....	231
Expand specific member(s)	232
Event	234
Data Shaping	240
Aggregation in React Pivotview component.....	240
Calculated field in React Pivotview component	253
Grouping in React Pivotview component	264
Filtering in React Pivotview component	264
Sorting in React Pivotview component.....	264
Drill through in React Pivotview component.....	264
Editing in React Pivotview component	264
Data Formatting	264
Number formatting in React Pivotview component.....	264
Conditional formatting in React Pivotview component.....	265
Report Manipulation.....	265
Grouping bar in React Pivotview component.....	265
Field list in React Pivotview component	265
Defer update in React Pivotview component.....	265
Performance	269
Virtual scrolling in React Pivot Table component.....	269
Paging in React Pivotview component.....	279
Data Compression in React Pivot Table component.....	295
State persistence in React Pivotview component	297
Save and Load Pivot Layout	299
Row and column in React Pivotview component	300
Width And Height.....	300
Row Height.....	302
Column Width	303
Reorder	306
Column Resizing	307
Text Wrap.....	308
Text Align	309
AutoFit.....	312
Grid Lines	316
Selection.....	318

Clip Mode	327
Cell Template	328
Events.....	333
See Also	339
Show hide totals in React Pivotview component	339
Show or hide grand totals	339
Show grand totals at top or bottom	340
Show or hide sub-totals	341
Show or hide sub-totals for specific fields	343
Show or hide totals using toolbar	344
Hyper link in React Pivotview component	345
Hyperlink for all cells.....	346
Hyperlink for row headers	347
Hyperlink for column headers.....	348
Hyperlink for value cells.....	349
Hyperlink for summary cells	350
Condition based hyperlink	352
Header based hyperlink	353
Event	354
See Also	356
Tool bar in React Pivotview component	356
Built-in toolbar options	356
Show desired chart types in the dropdown menu	362
Switch the chart to multiple axes	363
Show or hide legend	364
Adding custom option to the toolbar	365
Save and load report as a JSON file.....	370
Save and load reports to a SQL database	372
Events.....	395
See Also	412
Tool tip in React Pivotview component	413
Tooltip Template.....	414
Css customization in React Pivotview component	416
Hiding Axis.....	416
Text Alignment.....	419

Customize header, value and summary cell style.....	420
Printing and Exporting	422
Print in React Pivotview component.....	422
Excel export in React Pivot Table component	426
Pdf export in React Pivotview component	448
Globalization and localization in React Pivotview component.....	487
Load CLDR-Data to the application	487
Internationalization.....	489
Localization	495
Right-to-left (RTL).....	502
See Also	505
Accessibility in React Pivotview component.....	505
WAI-ARIA attributes.....	506
Keyboard interaction	508
Ensuring accessibility	514
See also	525
Ej1 api migration in React Pivotview component.....	525
Data Binding.....	525
Aggregation.....	526
Number Format.....	526
Summary Customization	526
Drill operation	527
Field List	528
Grouping Bar	528
Filtering	529
Maximum node limit in member editor	529
No Data Items	529
Excel-like filtering.....	529
Drill Through	530
Cell Editing	530
Hyperlink.....	530
Defer Layout Update.....	532
Sorting.....	532
Value Sorting.....	533
Calculated Field.....	533

Paging.....	533
Conditional Formatting	534
Exporting	534
Grid Customization	534
Accessibility	535
Common.....	536
How To	537
Switching older themes style in React Pivotview component.....	537
Customize number date and time values in React Pivotview component	540
Customize the icons for pivot table in React Pivotview component	542
PivotView_PivotFieldList .e-icons.e-toggle-field-list::before {	542
PivotView_PivotFieldList.e - icons.e - toggle - field - list;	543
Configure data grid options on editing mode in React Pivotview component.....	544
Refresh the field list in React Pivotview component	546
Hide empty headers in React Pivotview component.....	548
Customizing loading indicator in React Pivotview component	549
Changing the pivotview component minimum height in React Pivotview component	550
Chart based on pivot table selection in React Pivotview component	551
Drill through grid cell edit type in React Pivotview component	556
Show field list when pivot table empty in React Pivotview component	558
Apply custom style to pivot cells in React Pivotview component	559
Show tooltip for row and column headers in React Pivotview component	562
Hide specific columns in React Pivotview component	565
Export table and chart into the same document using toolbar in React Pivotview component	567
Add custom aggregation type to the menu in React Pivotview component.....	568
Convert complex JSON to flat JSON and assign it to the pivot table in React Pivotview component	571
Load desired report from the report list as default in React Pivotview component.....	578
Display string value to pivot table values in React Pivotview component	584
Summary of Predefined Dialogs component.....	586
Predefined dialogs	586
Getting Started.....	586
Dependencies.....	586
Setup your development environment	586
Adding Syncfusion packages	587

Adding CSS reference.....	587
Render a dialog using utility functions.....	587
Adding predefined dialogs to the application.....	588
Run the application	591
Alert dialog.....	593
Confirm dialog.....	596
Prompt dialog.....	599
Draggable in React Predefined dialogs component.....	603
Alert dragging.....	603
Confirm drag	605
Prompt drag	606
Animation in React Predefined dialogs component	608
Alert animation	609
Prompt animation	612
Position in React Predefined dialogs component.....	614
Alert position.....	615
Confirm position	617
Prompt position	618
Dimension in React Predefined dialogs component.....	620
Alert dimension.....	620
Confirm dimension.....	622
Prompt dimension	624
Max-width and max-height.....	626
Min-width and min-height	629
Customization in React Predefined dialogs component.....	631
Alert action button.....	631
Confirm action buttons	633
Prompt action buttons.....	635
Show or hide dialog close button	637
Alert dialog close button.....	637
Confirm dialog close button.....	639
Prompt dialog close button.....	641
Customize dialog content	643
Progress bar	645
Getting Started.....	645

Dependencies.....	645
Installation and configuration.....	645
Add Progressbar to the Project.....	646
Run the application	646
Types in React Progress bar component	647
Linear.....	647
Circular	648
Tooltip in React Progress bar component	649
Tooltip	649
Format.....	650
Customization	651
States in React Progress bar component.....	652
Determinate	652
Indeterminate	653
Buffer	654
Customization in React Progress bar component.....	655
Segments.....	655
Thickness.....	656
Radius.....	657
InnerRadius	658
Progress color and track color	659
Annotation in React Progress bar component.....	660
Annotation	660
Label.....	661
Animation in React Progress bar component	662
Range in React Progress bar component.....	663
Events in React Progress bar component	664
valueChanged.....	664
progressCompleted.....	665
Accessibility in React Progress bar component	666
WAI-ARIA attributes.....	667
Keyboard interaction	667
Ensuring accessibility	667
See also	668
ProgressButton	668

Getting Started.....	668
Dependencies.....	668
Installation and configuration.....	668
Adding syncfusion packages	669
Adding CSS reference.....	669
Adding ProgressButton component.....	669
Run the application	670
See Also	670
Spinner and progress in React Progress button component	671
Progress.....	671
See Also	676
Accessibility in React ProgressButton component	676
WAI-ARIA attributes.....	677
Keyboard interaction	677
Ensuring accessibility	677
See also	677
Style and appearance in React Progress button component	678
See also	678
How To	678
Change the text content and styles of the progressbutton during progress in React Progress button component.....	678
Customize progress using cssclass in React Progress button component.....	679
Hide spinner in React Progress button component.....	680
Enable progress in button in React Progress button component.....	681
Trace events of progress button in React Progress button component.....	681
Query Builder	683
Getting Started.....	683
Dependencies.....	683
Installation and Configuration	684
Adding Syncfusion packages	684
Adding CSS Reference	685
Adding Query Builder component to the Application	685
Run the application	686
Columns in React Query builder component.....	687
Auto generation	687

Labels	688
Operators	688
Step	688
Format.....	688
Validations	690
Data binding in React Query builder component	692
Local data	692
Remote data.....	694
Data Manager	702
Complex Data Binding.....	705
Filtering in React Query builder component	709
Templates in React Query builder component.....	712
Header Template	712
Column Template.....	717
Rule Template	724
Model binding in React Query builder component	731
Importing and Exporting in React Query builder component	733
Importing	733
Exporting.....	746
Lock Group/Rule in React Query builder component.....	759
Clone Group/Rule in React Query builder component.....	762
Global local in React Query builder component.....	765
Style and appearance in React Query builder component	768
Accessibility in React Query Builder component	768
WAI-ARIA attributes.....	769
Keyboard interaction	770
Ensuring accessibility	770
See also	770
How To	770
Render with rule in React Query builder component.....	770
Display mode in React Query builder component.....	772
Sort columns in React Query builder component.....	773
Restrict groups in React Query builder component	775
State persistence in React Query builder component.....	777
Rtl in React Query builder component	778

Summary view in React Query builder component	780
RadioButton	783
Getting Started.....	783
Dependencies.....	783
Installation and Configuration	783
Adding Syncfusion packages	783
Adding CSS Reference	784
Adding RadioButton component to the Application	784
Run the application	784
Label and size in React Radio button component	785
Label.....	785
Size	786
See Also	787
Accessibility in React RadioButton component	787
WAI-ARIA attributes.....	788
Keyboard interaction	788
Ensuring accessibility	788
See also	788
Style and appearance in React Radio button component	789
How To	789
Change radiobutton state in React Radio button component	789
Customize radiobutton appearance in React Radio button component.....	790
Name and value in form submit in React Radio button component	791
Right to left in React Radio button component.....	792
Set the disabled state in React Radio button component	793
Ej1 api migration in React Radio button component.....	795
Properties.....	795
Methods	796
Events.....	796
Range Navigator.....	797
Getting Started.....	797
Dependencies.....	797
Installation and configuration.....	797
Add Range Navigator to the Project	798
Module Injection.....	799

Populate Range Navigator with Data.....	800
Enable Tooltip	801
Selecting range in React Range navigator component	802
Lightweight in React Range navigator component.....	803
See Also	804
Series types in React Range navigator component	804
Line	804
Area	805
StepLine.....	806
Data in React Range navigator component	807
Numeric.....	807
Logarithmic Axis	812
Date-time	817
Period selector in React Range navigator component	820
Periods	821
Positioning period selector	822
Height.....	824
Visibility of range navigator	826
See Also	827
Labels in React Range navigator component.....	827
Multilevel labels	827
Grouping	828
Smart labels.....	830
Label positioning.....	831
Labels customization.....	832
Grid tick in React Range navigator component	833
Grid line customization	833
Tick line customization.....	834
Customization in React Range navigator component.....	836
Navigator appearance.....	836
Thumb	837
Border customization.....	838
Deferred update.....	839
Allow snapping.....	841
Animation.....	842

See Also	843
Tool tip in React Range navigator component.....	843
Customization	843
Label Format	844
R t l in React Range navigator component.....	846
Export print in React Range navigator component	847
Export.....	847
Print.....	848
Accessibility in React Range navigator component	850
WAI-ARIA attributes.....	851
Keyboard interaction	851
Ensuring accessibility	851
See also	851
Ej1 api migration in React Range navigator component	851
RangeNavigator.....	851
Series.....	854
StyleSettings.....	855
Tooltip	857
Period Selector.....	858
Methods.....	858
Events.....	858
Range Slider	859
Getting Started.....	859
Dependencies.....	860
Installation and Configuration	860
Adding Syncfusion packages	860
Adding CSS Reference	861
Adding Slider component	861
Run the Application.....	862
Types	862
Customization	864
See Also	866
Ticks in React Range slider component	866
Step	867
Min and Max	868

Format in React Range slider component.....	870
Using format API	871
Using Events	872
Limits in React Range slider component.....	874
Default and MinRange Slider limits	874
Range Slider limits.....	875
Handle lock.....	877
Style in React Range slider component	878
Customizing the slider track.....	878
Customizing the slider handle.....	878
Customizing the slider limits	878
Customizing the slider ticks	878
Customizing the slider buttons	879
Accessibility in React Range Slider component	879
WAI-ARIA attributes.....	880
Keyboard interaction	880
Ensuring accessibility	881
See also	881
Ej1 api migration in React Range slider component.....	881
How To	883
Time range slider in React Range slider component	883
Date range slider in React Range slider component.....	884
Numeric range slider in React Range slider component.....	886
Customize slider bar in React Range slider component	888
Customize slider limits in React Range slider component	892
Customize slider ticks label in React Range slider component.....	894
ticks_slider .e-scale :nth-child(1)::before {	894
Customize slider thumb in React Range slider component	896
square_slider.e-control.e-slider .e-handle {	896
circle_slider.e-control.e-slider .e-handle {	897
oval_slider.e-control.e-slider .e-handle {	897
Form slider with formvalidator in React Range slider component.....	898
Show slider from hidden state in React Range slider component.....	907
Reversible Range Slider in React.....	909
Rating	911

Getting Started with React Rating Control	911
Dependencies.....	912
Installation and Configuration	912
Adding Syncfusion packages	912
Adding Rating component to the Application	912
Adding CSS Reference	913
Running the application	913
Value	914
Precision Modes in React Rating Component.....	915
Labels in React Rating Component	916
Label position	917
Label template	918
Tooltip in React Rating Component.....	919
Tooltip template	919
Tooltip customization	921
Selection in React Rating Component	922
Min value	923
Single selection	923
Show or hide reset button	924
Templates in React Rating Component	925
Empty (unrated) symbol template.....	925
Full (rated) symbol template	928
Using Emoji icon as rating symbol	930
Using SVG icon as rating symbol.....	932
Using PNG image as rating symbol	934
Events in React Rating Component.....	935
beforeItemRender	935
created	936
onItemHover	937
valueChanged.....	937
Appearance in React Rating Component.....	939
Items Count.....	939
Disabled.....	939
Visible.....	940
CssClass	942

Changing icon using CssClass	945
Accessibility in React Rating component	947
WAI-ARIA attributes	948
Keyboard interaction	949
Ensuring accessibility	949
See also	949
Ribbon	949
Getting Started.....	949
Dependencies.....	949
Setup your development environment	950
Adding Syncfusion packages	951
Adding Style sheet to the Application.....	951
Add Ribbon to the project.....	951
Adding Ribbon Tab	952
Adding Ribbon Group.....	953
Adding Ribbon Item	954
Run the application	955
Modules in Ribbon component	964
Tabs and Groups	964
Adding Tabs.....	965
Adding Groups	966
Adding Items	967
Ribbon Items	969
Built-in items.....	969
Custom items	999
Items display Mode.....	1001
Enable or disable items	1005
Ribbon Layouts.....	1007
Classic layout.....	1007
Simplified layout	1033
Minimized State	1041
File Menu	1044
Visibility.....	1044
Adding menu items	1046
Open submenu on click.....	1048

Custom header text	1050
Ribbon Backstage	1052
Adding backstage items	1052
Adding footer items	1057
Adding separator.....	1062
Back button.....	1067
Backstage target.....	1070
Template	1075
Setting width and height.....	1081
Ribbon contextual tabs	1085
Visible tabs	1085
Adding contextual tabs	1085
Selected tabs.....	1089
Methods.....	1092
Ribbon Keytips in React Ribbon component	1096
Ribbon items keytip	1096
File menu keytip.....	1103
Backstage menu keytip	1106
Ribbon layout switcher keytip	1110
Ribbon launcher icon keytip	1113
Methods.....	1116
Guidelines for adding keytips.....	1116
Gallery Items in React Ribbon component	1116
Groups.....	1116
Setting item count.....	1147
Setting selected item	1150
Setting popup height.....	1154
Setting popup width.....	1154
Help Pane	1157
Tooltip	1159
Adding Title	1159
Adding Content	1162
Adding Icon	1164
Customization	1167
Ribbon Resizing	1171

Defining items allowed size	1171
Defining items active size.....	1172
Events.....	1172
tabSelected	1173
tabSelecting.....	1174
ribbonCollapsing	1175
ribbonExpanding	1176
launcherIconClick	1177
Button item events	1178
Checkbox item events	1180
Colorpicker item events	1183
ComboBox item events	1191
DropDown item events	1199
SplitButton item events	1208
GroupButton item events	1217
FileMenu events.....	1220
Backstage view events	1229
Gallery events	1231
Accessibility in React Ribbon component.....	1241
WAI-ARIA attributes.....	1242
Keyboard interaction	1244
Ensuring accessibility	1245
See also	1245
RichTextEditor	1245
Getting Started with React Rich Text Editor Component	1245
Prerequisites	1245
Dependencies.....	1245
Create the React application.....	1246
Add Syncfusion React packages	1247
Import the Syncfusion CSS styles	1247
Add Rich Text Editor component to the application.....	1248
Module Injection.....	1251
Run the application	1252
Configure the Toolbar	1255
Retrieve the formatted content.....	1257

Insert images and links.....	1257
Editor modes in React Rich text editor component.....	1260
HTML editor	1260
Markdown editor	1265
Toolbar in React Rich text editor component.....	1273
Expand Toolbar	1273
MultiRow toolbar	1278
Floating toolbar.....	1284
Toolbar items	1289
Custom tool.....	1294
Quick inline toolbar	1306
Styling in React Rich text editor component	1314
Font name and size	1314
Custom font and size	1320
Font and Background color	1326
Editor content styles	1331
Image in React Rich text editor component	1337
Upload options.....	1337
Image delete	1342
Insert from web	1348
Dimension	1349
Caption and Alt Text.....	1350
Display position.....	1350
Image with link.....	1350
Resize	1351
Drag and Drop.....	1351
Audio in React Rich text editor component.....	1357
Configure audio tool in the toolbar	1357
Insert audio from the web	1362
Insert audio from local machine	1363
Replacing audio.....	1369
Delete audio.....	1369
Display position.....	1370
Rename audio before inserting	1371
Upload audio with authentication	1375

See Also	1377
Video in React Rich text editor component.....	1377
Configure the video tool in the toolbar	1378
Insert a video from the web.....	1382
Insert video from local machine	1383
Replacing video	1390
Delete video	1390
Dimension	1391
Display position.....	1392
Resize video.....	1393
Rename video before inserting.....	1393
Upload video with authentication	1397
See Also	1399
Link in React Rich text editor component.....	1399
Insert link	1399
Remove link.....	1405
Auto-link.....	1405
Manipulation.....	1405
Table in React Rich text editor component	1410
Insert table	1410
Quick Toolbar	1414
Table Header.....	1415
Insert Rows.....	1415
Insert Columns	1415
Set Color.....	1415
Delete Table	1415
Horizontal Align.....	1416
Table Styles	1416
Table Properties	1416
Table cell merge and split	1417
Emoji Picker in React RichTextEditor Component	1422
Enabling the toolbar option and custom emojis.....	1422
Using the shortcut key to open the emoji picker.....	1426
Navigating and selecting emojis using the keyboard.....	1427
Markdown in React Rich text editor component.....	1427

Supported Commands	1427
Markdown to HTML	1429
Table.....	1442
Custom format	1468
File browser in React Rich text editor component	1476
Required additional dependency.....	1476
Additional CSS Reference.....	1477
Format Painter in React Rich Text Editor Component Syncfusion.....	1483
Enabling the toolbar option for Format Painter	1483
Customization of copy and paste format.....	1488
Using the shortcut key to copy and paste the format	1494
Form support in React Rich text editor component	1495
Validation in React Rich text editor component.....	1499
Validation rules	1499
Validation message	1500
Custom placement of validation message	1504
Globalization in React Rich text editor component.....	1510
Localization	1510
RTL.....	1530
Miscellaneous in React Rich text editor component	1535
Placeholder	1535
Character count	1536
Code view.....	1542
Undo/Redo manager.....	1554
Prevention of cross-site scripting (XSS)	1559
Resizable support.....	1564
Number and Bullet Format Lists	1568
Xhtml validation in React Rich text editor component.....	1572
Attributes	1572
HTML Elements	1573
Inline mode in React Rich text editor component.....	1577
Show on select/click.....	1577
Paste cleanup in React Rich text editor component.....	1582
MS Word to HTML	1582
Paste cleanup	1582

Prompt dialog.....	1583
Paste as plain text	1583
Keep format	1583
Denied tags	1584
Denied attributes	1584
Allowed style properties	1584
Mention integration in React Rich text editor component	1589
See Also	1597
Enter key in React Rich text editor component	1597
Enter key customization.....	1597
Shift-Enter key customization.....	1604
Iframe in React Rich text editor component.....	1610
IFrame attributes	1615
Adding external CSS/Script file	1619
Third party integration in React Rich text editor component	1624
CodeMirror Integration.....	1624
Embed.ly Integration.....	1637
Exec command in React Rich text editor component.....	1637
Style in React Rich text editor component	1639
Customizing the Rich Text Editor's content	1639
Customizing the Rich Text Editor's toolbar	1639
Customizing the Rich Text Editor's character count	1640
Keyboard support in React Rich text editor component	1640
HTML formation shortcut key.....	1640
Markdown formation shortcut key.....	1647
Custom key config.....	1652
Accessibility in React Rich text editor component.....	1658
ARIA attributes.....	1659
Keyboard interaction	1664
Ensuring accessibility	1666
See also	1666
How To	1666
Add google fonts in React Rich text editor component.....	1666
Change default font family in React Rich text editor component	1671
Check image size in React Rich text editor component.....	1676

Customize placeholder style in React Rich text editor component	1679
Customize shortcut keys in React Rich text editor component	1680
Set the cursor at the specific range in React Rich text editor component	1684
Update value in React Rich text editor component.....	1687
Rename images in server in React Rich text editor component.....	1690
File attachment in React Rich text editor component.....	1695
Format code block in React Rich text editor component	1700
Schedule	1703
Getting Started.....	1703
Getting Started.....	1703
Creating a Next.js Application Using Syncfusion React Components	1714
Getting Started with the React Schedule Component in the Preact Framework.....	1719
Module injection in React Schedule component.....	1723
Module injection.....	1723
Scheduler interactions in React Schedule component	1724
Appointments in React Schedule component	1725
Normal events.....	1725
Spanned events.....	1727
All-day events.....	1727
Expand all day appointments view on initial load	1728
Customize the rendering of the spanned events.....	1732
Recurring events	1734
Event fields.....	1747
Adding Custom fields	1754
Customize the order of the overlapping events	1756
Drag and drop appointments.....	1759
Inline Appointment	1783
Appointment Resizing	1786
Appointment customization	1796
Setting minimum height	1803
Block Dates and Times	1806
Readonly	1810
Make specific events readonly.....	1812
Restricting event creation on specific time slots	1814
Differentiate the past time events.....	1817

Appointments occupying entire cell	1819
How to limit maximum number of events to display	1821
Display tooltip for appointments	1823
Appointment selection	1827
Retrieve event details from the UI of an event	1828
Get the current view appointments	1831
Get the entire appointment collections.....	1835
Refresh appointments	1838
Data binding in React Schedule component.....	1839
Binding local data.....	1839
Binding remote data	1841
Loading data via AJAX post	1850
Passing additional parameters to the server	1851
Handling failure actions	1854
Scheduler CRUD actions.....	1856
Configuring Scheduler with Google API service	1860
Crud actions in React Schedule component	1863
Add	1863
Edit	1873
Delete.....	1889
Drag and drop	1900
Resize	1903
Virtual scrolling in React Schedule component	1905
Enabling lazy loading for appointments.....	1910
See Also	1914
Editor template in React Schedule component.....	1914
Event editor.....	1914
Customizing event editor using template.....	1932
Quick popups	1959
More events indicator and popup	1971
Timezone in React Schedule component.....	1987
Understanding date manipulation in JavaScript.....	1988
Scheduler with no timezone	1988
Scheduler set to specific timezone	1990
Display events on same time everywhere with no time difference	1991

Set specific timezone for events	1994
Add or remove timezone names to/from the timezone collection	1996
Timezone methods	1998
Views in React Schedule component	2000
Setting specific view on scheduler	2000
View specific configuration	2005
Extending view intervals	2034
See Also	2036
Calendar mode in React Schedule component	2036
Gregorian Calendar	2036
Islamic Calendar	2037
Resources in React Schedule component	2039
Resource fields	2040
Resource data binding	2041
Scheduler with multiple resources	2042
Resource grouping	2045
Working with shared events	2061
Simple resource header customization	2065
Customizing resource header with multiple columns	2069
Collapse/Expand child resources in timeline views	2072
Displaying tooltip for resource headers	2075
Choosing between resource colors for appointments	2078
Dynamically add and remove resources	2082
Setting different working days and hours for resources	2086
Hide non-working days when grouped by date	2091
Compact view in mobile	2093
Adaptive UI in desktop	2095
Header rows in React Schedule component	2098
Display year and month rows in timeline views	2101
Display week numbers in timeline views	2103
Timeline view displaying dates of a complete year	2105
Customizing the header rows using template	2107
Row auto height in React Schedule component	2110
Calendar month view	2110
Timeline views	2112

Timeline views with multiple resources	2114
Appointments occupying entire cell	2116
Header bar in React Schedule component	2119
Show or Hide header bar	2119
Customizing header bar	2121
How to display the view options within the header bar popup	2125
Date header customization.....	2127
Customizing the date range text.....	2134
Customizing header indent cells	2136
Timescale in React Schedule component	2139
Setting different time slot duration	2140
Customizing time cells using template	2142
Hide the timescale	2144
Highlighting current date and time.....	2146
Working days in React Schedule component	2148
Set working days	2148
Hiding weekend days	2150
Show week numbers.....	2152
Set working hours	2156
Scheduler displaying custom hours	2158
Setting start day of the week	2160
Scroll to specific time and date.....	2162
See Also	2167
Cell customization in React Schedule component.....	2167
Setting cell dimensions in all views.....	2167
Check for cell availability.....	2170
Customizing cells in all the views	2172
Customizing cell header in month view	2180
Customizing the minimum and maximum date values	2183
Customizing the weekend cells background color.....	2186
How to disable multiple cell and row selection in Schedule	2189
State persistence in React Schedule component.....	2189
Exporting in React Schedule component.....	2191
Excel Exporting	2191
Exporting calendar events as ICS file	2213

Import events from other calendars.....	2221
How to print the Scheduler element	2225
Context menu in React Schedule component	2231
Dimensions in React Schedule component.....	2239
Auto Height and Width	2239
Height and Width in pixel	2241
Height and Width in percentage.....	2243
See Also	2245
Recurrence editor in React Schedule component	2245
Customizing the repeat type option in editor	2245
Customizing the End Type Option in Editor	2248
Accessing the recurrence rule string.....	2250
Set specific value on recurrence editor	2252
Recurrence date generation	2254
Recurrence date generation in server-side.....	2257
Restrict date generation with specific count	2257
Localization in React Schedule component	2260
Globalization	2260
Localizing the static Scheduler text.....	2264
Setting date format.....	2270
Setting the time format	2272
First day of the week.....	2274
Displaying Scheduler in RTL mode	2276
See Also	2278
Accessibility in React Schedule component.....	2278
ARIA attributes.....	2279
Keyboard interaction	2279
Ensuring accessibility	2280
See also	2280
Scheduler styling in React Schedule component.....	2280
Frequently asked questions in React Schedule component	2282
Maximum call stack size exceeded	2282
Grouping with empty resources	2283
Not providing e-field in editor template.....	2284
Missing CSS reference	2284

QuickInfoTemplate at bottom	2285
Not importing culture files while using localization	2285
Ej1 api migration in React Schedule component	2286
Scheduler	2286
Properties.....	2286
Methods.....	2297
Events.....	2300
How To	2303
Add edit and remove events in React Schedule component.....	2303
Set default value for event fields in React Schedule component.....	2311
Open event editor manually in React Schedule component	2313
Prevent date navigation in React Schedule component.....	2319
Half yearly view in React Schedule component.....	2321
Set different event time duration in React Schedule component.....	2325
Set different work hours in React Schedule component	2326
Show quick info template in React Schedule component	2329
Enable scroll option on all day section in React Schedule component	2337
Manual refresh in React Schedule component.....	2339
Sidebar	2351
Getting Started.....	2351
Dependencies.....	2351
Installation and configuration.....	2352
Adding Syncfusion packages	2352
Adding Style sheet to the Application.....	2352
Adding Sidebar component to the Application	2353
Run the application	2354
Enable backdrop	2355
Position	2358
Animate.....	2361
Close on document click	2363
Enable gestures.....	2365
See Also	2367
Custom context in React Sidebar component	2367
Variations in React Sidebar component	2370
Expanding types of Sidebar.....	2370

See Also	2375
Auto close in React Sidebar component	2375
Docking sidebar in React Sidebar component	2378
See Also	2381
Style in React Sidebar component	2381
Customizing the sidebar	2381
Customizing the sidebar based on the positions	2381
Customizing the sidebar based on the active state	2382
Customizing the sidebar with dock state	2382
Customizing the different types of sidebar	2383
Customizing the backdrop of the sidebar	2383
Customizing the sidebar in the RTL direction	2384
Prevent the animation transition for the Sidebar component	2384
Accessibility in React Sidebar component	2384
WAI-ARIA attributes	2385
Keyboard interaction	2385
Ensuring accessibility	2385
See also	2385
How To	2386
Sidebar with listview in React Sidebar component	2386
Open close sidebar in React Sidebar component	2388
Multiple sidebar in React Sidebar component	2391
Sidebar with treeview in React Sidebar component	2393
Fixed sidebar in React Sidebar component	2401
Sidebar with variation animation in React Sidebar component	2411

PivotTable

Getting started

This section explain steps to create a simple [Link to the Video](#) with OLAP data source in React environment.

To get start quickly with React Pivot Table, you can check on this video:

Dependencies

The following list of dependencies are required to use the pivot table component in your application.

```
`javascript
|-- @syncfusion/ej2-react-pivotview
|-- @syncfusion/ej2-pivotview
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-excel-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-pdf-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-charts
|-- @syncfusion/ej2-svg-base
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-grids
|-- @syncfusion/ej2-react-base
`,`
```

Setup for Local Development

You can use [create-react-app](#) to setup the application. To install **create-react-app** run the following command.

```
`
```

```
npm install -g create-react-app
```

,

To create basic **React** application use following commands.

```
<div class='tsx'>
```

,

```
create-react-app quickstart --scripts-version=react-scripts-ts
```

,

Now, the application is created in the **quickstart** demo folder. Run the following command one-by-one to navigate to the **quickstart** demo folder, and install the required **npm** dependent packages.

,

```
cd quickstart
```

```
npm install
```

,

```
</div>
```

```
<div class='jsx'>
```

,

```
create-react-app quickstart
```

,

Now, the application is cloned in the **quickstart** demo folder. Run the following command one-by-one to navigate to the **quickstart** demo folder, and install the required **npm** dependent packages.

,

```
cd quickstart
```

```
npm install
```

,

```
</div>
```

Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) public registry. To install pivot table component, use the following command.

,

```
npm install @syncfusion/ej2-react-pivotview --save
```

,

The **--save** will instruct NPM to include the pivot table package inside the **dependencies** section of the **package.json**.

Adding CSS reference

Add pivot table and its [dependent](#) components styles as given below in **src/App.css** file. In this illustration, we have referred **material** theme.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-react-pivotview/styles/material.css';
`
```

You can also refer other themes like bootstrap, fabric, high-contrast etc. To know about individual component CSS, please refer [here](#).

Next we need to refer **App.css** in the application by importing it in the **src/App.tsx** file as follows.

```
`ts
import './App.css';
`
```

Browser compatibility

Polyfills are required to use the Pivot Table in Internet Explorer 11 browser. Refer the [documentation](#) for more details.

Adding pivot table component

You can initialize pivot table component in the application using following steps.

- Import the **PivotViewComponent** (aka, PivotTable) component from the **@syncfusion/ej2-react-pivotview** package in **app.ts** file.
- Then you can initialize pivot table component () using following code.

```
`ts
import { IDataOptions, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
```

```

return (<PivotViewComponent/>)
};
export default App;
`

```

After initialization, add the the following code in **src/App.tsx** file to populate pivot table with a sample relational data source. Refer [here](#) to know the more details about relational data binding.

```

`ts
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from "react-dom";
let pivotData = [
  { 'Sold': 31, 'Amount': 52824, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q1' },
  { 'Sold': 51, 'Amount': 86904, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q2' },
  { 'Sold': 90, 'Amount': 153360, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q3' },
  { 'Sold': 25, 'Amount': 42600, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q4' },
  { 'Sold': 27, 'Amount': 46008, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2016', 'Quarter': 'Q1' }];
function App() {
  const dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'CO' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent id='PivotView' height={350}
    dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};

```

```
export default App;

ReactDOM.render(<App />, document.getElementById("sample"));
`ts

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from "react-dom";

let pivotData = [
  { 'Sold': 31, 'Amount': 52824, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q1' },
  { 'Sold': 51, 'Amount': 86904, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q2' },
  { 'Sold': 90, 'Amount': 153360, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q3' },
  { 'Sold': 25, 'Amount': 42600, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q4' },
  { 'Sold': 27, 'Amount': 46008, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2016', 'Quarter': 'Q1' }
];

function App() {
  const dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
  };

  let pivotObj;

  return (<PivotViewComponent id='PivotView' height={350}
    dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}

export default App;
```

```
ReactDOM.render(<App />, document.getElementById("sample"));
```

Adding fields to row, column, value and filter axes

Now that pivot table is initialized and assigned with sample data, will further move to showcase the component by organizing appropriate fields in row, column, value and filter axes.

In [dataSourceSettings](#), four major axes - [rows](#), [columns](#), [values](#) and [filters](#) plays a vital role in defining and organizing fields from the bound data source, to render the entire pivot table component in a desired format.

[rows](#) – Collection of fields that needs to be displayed in row axis of the pivot table.

[columns](#) – Collection of fields that needs to be displayed in column axis of the pivot table.

[values](#) – Collection of fields that needs to be displayed as aggregated numeric values in the pivot table.

[filters](#) - Collection of fields that would act as master filter over the data bound in row, column and value axes of the pivot table.

In-order to define each field in the respective axis, the following basic properties should be set.

- [name](#): It allows to set the field name from the bound data source. It's casing should match exactly like in the data source and if not set properly, the pivot table will not be rendered.
- [caption](#): It allows to set the field caption, which is the alias name of the field that needs to be displayed in the pivot table.
- [type](#): It allows to set the summary type of the field. By default, **Sum** is applied.

In this illustration, "Year" and "Quarter" are added in column, "Country" and "Products" in row, and "Sold" and "Amount" in value section respectively.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  const dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }],
  };
  let pivotObj;
  return (
    <PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings}>
      </PivotViewComponent>
    );
}
```



```
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    const dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}>
        </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));
```

Applying formatting to a value field

Formatting defines a way in which values should be displayed. For example, format “C” denotes the values should be displayed in currency pattern. To do so, define the [formatSettings](#) with its [name](#) and [format](#) properties and add it to [formatSettings](#). In this illustration, the [name](#) property is set as **Amount**, a field from value section and its format is set as currency. Likewise, we can set format for other value fields as well and add it to [formatSettings](#).

Only fields from value section, which is in the form of numeric data values are applicable for formatting.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    const dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
    };
    return (<PivotViewComponent id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}>
        </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));
```

```

        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}>
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    const dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}>
    </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));

```

Module Injection

To create pivot table with additional features, inject the required modules. The modules that are available with basic functionality are as follows.

- **GroupingBar** - Inject this module to access grouping bar.
- **FieldList** - Inject this module to access pivot field list.
- **CalculatedField** - Inject this module to access calculated field.

These modules should be injected into the PivotView using the **Inject** method within the **app.tsx** file as shown below. On doing so, only the injected views will be loaded and displayed along with pivot table.

`ts

```
<Inject services={[GroupingBar]} />
```

Enable Field List

The field list allows to add or remove fields and also rearrange the fields between different axes, including column, row, value, and filter along with filter and sort options dynamically at runtime. It can be enabled by setting the [showFieldList](#) property to **true** and by injecting the **FieldList** module as follows. To know more about field list, [refer](#) here.

If the **FieldList** module is not injected, the Field List will not be rendered with the pivot table component.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    const dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]} />
</PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    const dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]} />
</PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));
```

```

    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]} />
  </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));

```

Enable Grouping Bar

The grouping bar feature automatically populates fields from the bound data source and allows end users to drag fields between different axes such as columns, rows, values, and filters, and alter pivot table at runtime. It also provides option to sort, filter and remove fields. It can be enabled by setting the [showGroupingBar](#) property to **true** and by injecting the **GroupingBar** module as follows. To know more about grouping bar, [refer](#) here.

If the **GroupingBar** module is not injected, the grouping bar will not be rendered with the pivot table component.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { Inject, PivotViewComponent, GroupingBar } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  const dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<PivotViewComponent id='PivotView' height={350}
dataSourceSettings={dataSourceSettings} showGroupingBar={true}><Inject
services={[GroupingBar]} />
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));

```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { IDataOptions, IDataset, Inject, PivotViewComponent, GroupingBar }
from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    const dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent id='PivotView' height={350}
dataSourceSettings={dataSourceSettings} showGroupingBar={true}><Inject
services={[GroupingBar]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));
```

Exploring Filter Axis

The filter axis contains collection of fields that would act as master filter over the data bound in [rows](#), [columns](#) and [values](#) axes of the pivot table. The fields along with filter members could be set to filter axis either through report via code behind or by dragging and dropping fields from other axes to filter axis via grouping bar or field list at runtime.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { CalculatedField, FieldList, Inject, PivotViewComponent } from
'@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    const dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [{ name: 'Quarter' }],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}><Inject
services={[CalculatedField, FieldList]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));
```

```

    </PivotViewComponent>);
  }
;
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { CalculatedField, FieldList, IDataOptions, IDataset, Inject,
PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  const dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [{ name: 'Quarter' }],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' } ]
  };
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}><Inject
services={[CalculatedField, FieldList]}/>
    </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));

```

Calculated Field

The calculated field feature allows user to insert or add a new calculated field based on the available fields from the bound data source using basic arithmetic operators. The calculated field can be included in pivot table using the [CalculatedFieldSettings](#) from code behind. Or else, calculated fields can be added at run time through the built-in dialog by just setting the [allowCalculatedField](#) property to **true** and by injecting the **CalculatedField** module as follows in pivot table. You will see a button enabled in the Field List UI automatically to invoke the calculated field dialog and perform necessary operation. To know more about calculated field, [refer](#) here.

If the **CalculatedField** module is not injected, the calculated field dialog will not be rendered with the pivot table component. Moreover calculated measure can be added only in value axis.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { CalculatedField, FieldList, Inject, PivotViewComponent } from
 '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';

```

```
function App() {
  const dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }, { name: 'Total', caption: 'Total Units', type:
'CalculatedField' }],
    calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount)"+"Sum(Sold)"' }]
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}><Inject
services={[CalculatedField, FieldList]}/>
</PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { CalculatedField, FieldList, IDataOptions, IDataset, Inject,
PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  const dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }, { name: 'Total', caption: 'Total Units', type:
'CalculatedField' }],
    calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount)"+"Sum(Sold)"' }]
  };
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}><Inject
services={[CalculatedField, FieldList]}/>
</PivotViewComponent>);
```

```
};
export default App;
ReactDOM.render(<App />, document.getElementById("sample"));
```

Run the application

The quickstart project is configured to compile and run the application in the browser. Use the following command to run the application.

```
npm start
```

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { CalculatedField, FieldList, Inject, PivotViewComponent } from
 '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    const dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
 caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
 height={350} dataSourceSettings={dataSourceSettings}
 allowCalculatedField={true} showFieldList={true}><Inject
 services={[CalculatedField, FieldList]}/>
 </PivotViewComponent>);
}
;
export default App;
const root = ReactDOM.createRoot(document.getElementById('sample'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { CalculatedField, FieldList, IDataOptions, IDataset, Inject,
 PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    const dataSourceSettings: IDataOptions = {
```



```
columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
dataSource: pivotData as IDataset[],
expandAll: false,
filters: [],
drilledMembers: [{ name: 'Country', items: ['France'] }],
formatSettings: [{ name: 'Amount', format: 'C0' }],
rows: [{ name: 'Country' }, { name: 'Products' }],
values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
};
let pivotObj: PivotViewComponent;
return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}><Inject
services={[CalculatedField, FieldList]}/>
</PivotViewComponent>);
};
export default App;
const root = ReactDOM.createRoot(document.getElementById('sample'));
root.render(<App />);
```

You can also explore our [React Pivot Table example](#) that shows how to rendering of the pivot table with drill-up and drill-down functionality bound to a relational report.

Creating a Next.js Application Using Syncfusion React Components

This section provides a step-by-step guide for setting up a Next.js application and integrating the Syncfusion React Pivot Table component.

What is Next.js?

[Next.js](#) is a React framework that makes it easy to build fast, SEO-friendly, and user-friendly web applications. It provides features such as server-side rendering, automatic code splitting, routing, and API routes, making it an excellent choice for building modern web applications.

Prerequisites

Before getting started with the Next.js application, ensure the following prerequisites are met:

- [Node.js 16.8](#) or later.
- The application is compatible with macOS, Windows, and Linux operating systems.

Create a Next.js application

To create a new **Next.js** application, use one of the commands that are specific to either NPM or Yarn.

NPM

```
npx create-next-app@latest
```

YARN

```
yarn create next-app
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: Users can specify the name of the project directly. Let's specify the name of the project as `ej2-nextjs-pivotview`.

CMD

```
✓ What is your project named? » ej2-nextjs-pivotview
```

2. Select the required packages.

CMD

```
✓ What is your project named? ... ej2-nextjs-pivotview
✓ Would you like to use TypeScript? ... No / `Yes`
✓ Would you like to use ESLint? ... No / `Yes`
✓ Would you like to use Tailwind CSS? ... `No` / Yes
✓ Would you like to use `src/` directory? ... No / `Yes`
✓ Would you like to use App Router? (recommended) ... No / `Yes`
✓ Would you like to customize the default import alias? ... `No` / Yes
Creating a new Next.js app in D:\ej2-nextjs-pivotview.
```

3. Once complete the above mentioned steps to create `ej2-nextjs-pivotview`, navigate to the directory using the below command:

CMD

```
cd ej2-nextjs-pivotview
```

The application is ready to run with default settings. Now, let's add Syncfusion components to the project.

Install Syncfusion React packages

Syncfusion React component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion React components in the project, install the corresponding npm package.

Here, the [React Pivot Table component](#) is used in the project. To install the React Pivot Table component, use the following command:

NPM

```
npm install @syncfusion/ej2-react-pivotview --save
```

YARN

```
yarn add @syncfusion/ej2-react-pivotview
```

Import Syncfusion CSS styles

Syncfusion React components come with [built-in themes](#), which are available in the installed packages. It's easy to adapt the Syncfusion React components to match the style of your application by referring to one of the built-in themes.

Import the **Material** theme into the **src/app/globals.css** file and removed the existing styles in that file, as shown below:

GLOBALS.CSS

```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-react-pivotview/styles/material.css';
```

To know more about built-in themes and CSS reference for individual components, refer to the [themes](#) section.

Add Syncfusion React component

Follow the below steps to add the React Pivot Table component to the Next.js project:

1. Before adding the Pivot Table component to your markup, create a **datasource.tsx** file within the **src/app/** folder and add the Pivot Table component data.

DATASOURCE.TSX

```
export let pivotData: object[] = [
  { 'In_Stock': 42, 'Sold': 80, 'Amount': 2460, 'Country': 'Germany',
    'Product_Categories': 'Accessories', 'Products': 'Hydration Packs',
    'Order_Source': 'Retail Outlets', 'Year': 'FY 2015', 'Quarter': 'Q1' },
  { 'In_Stock': 19, 'Sold': 16, 'Amount': 184, 'Country': 'Germany',
    'Product_Categories': 'Accessories', 'Products': 'Fenders', 'Order_Source':
    'Retail Outlets', 'Year': 'FY 2015', 'Quarter': 'Q1' },
  { 'In_Stock': 36, 'Sold': 25, 'Amount': 188, 'Country': 'Germany',
    'Product_Categories': 'Accessories', 'Products': 'Bottles and Cages',
    'Order_Source': 'Retail Outlets', 'Year': 'FY 2015', 'Quarter': 'Q1' },
  { 'In_Stock': 6, 'Sold': 51, 'Amount': 229.5, 'Country': 'Germany',
    'Product_Categories': 'Accessories', 'Products': 'Cleaners', 'Order_Source':
    'Retail Outlets', 'Year': 'FY 2015', 'Quarter': 'Q1' },
  { 'In_Stock': 47, 'Sold': 337, 'Amount': 1110, 'Country': 'Germany',
    'Product_Categories': 'Clothing', 'Products': 'Vests', 'Order_Source': 'App
    Store', 'Year': 'FY 2017', 'Quarter': 'Q4' },
  { 'In_Stock': 14, 'Sold': 535, 'Amount': 10165, 'Country': 'United States',
    'Product_Categories': 'Accessories', 'Products': 'Tires and Tubes',
    'Order_Source': 'App Store', 'Year': 'FY 2017', 'Quarter': 'Q4' },
  { 'In_Stock': 47, 'Sold': 405, 'Amount': 3037.5, 'Country': 'United States',
    'Product_Categories': 'Accessories', 'Products': 'Bottles and Cages',
    'Order_Source': 'App Store', 'Year': 'FY 2017', 'Quarter': 'Q4' },
  { 'In_Stock': 14, 'Sold': 385, 'Amount': 11646.25, 'Country': 'United
    States', 'Product_Categories': 'Accessories', 'Products': 'Hydration Packs',
    'Order_Source': 'App Store', 'Year': 'FY 2017', 'Quarter': 'Q4' },
```

```
{ 'In_Stock': 11, 'Sold': 426, 'Amount': 1704, 'Country': 'United States',
'Product_Categories': 'Accessories', 'Products': 'Cleaners', 'Order_Source':
'App Store', 'Year': 'FY 2017', 'Quarter': 'Q4' },
{ 'In_Stock': 13, 'Sold': 392, 'Amount': 5989.76, 'Country': 'United
States', 'Product_Categories': 'Accessories', 'Products': 'Helmets',
'Order_Source': 'App Store', 'Year': 'FY 2017', 'Quarter': 'Q4' },
{ 'In_Stock': 9, 'Sold': 426, 'Amount': 4686, 'Country': 'United States',
'Product_Categories': 'Accessories', 'Products': 'Fenders', 'Order_Source':
'App Store', 'Year': 'FY 2017', 'Quarter': 'Q4' }
];
```

2. Then, import and define the Pivot Table component in the **src/app/page.tsx** file, as shown below:

PAGE.TSX

```
'use client'
import { CalculatedField, FieldList, IDataOptions, IDataset, Inject,
PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
export default function Home() {
  const dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter'
}],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['Germany'] }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption:
'Sold Amount' }]
  };
  return (
    <>
    <h2>Syncfusion React Pivot Table Component</h2>
    <PivotViewComponent id='PivotView' height={350}
dataSourceSettings={dataSourceSettings} allowCalculatedField={true}
showFieldList={true}><Inject services={[CalculatedField, FieldList]} />
    </PivotViewComponent>
    </>
  )
}
```

Run the application

To run the application, use the following command:

NPM

```
npm run dev
```

YARN

```
yarn run dev
```

To learn more about the functionality of the Pivot Table component, refer to the [documentation](#).

[View the NEXT.js Pivot Table sample in the GitHub repository](#)[Link to the Video](#).

Data binding in React Pivotview component

To get start quickly with Data Binding, you can check on this video:

JSON

For JSON data binding, the `type` property under [dataSourceSettings](#) needs to be set as `JSON`. By default, the default value is assumed as `JSON`.

Binding JSON data via local

In-order to bind local JSON data to the pivot table user can assign the local variable holding the JSON data to the [dataSource](#) property under [dataSourceSettings](#).

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
```

```

    drilledMembers: [{ name: 'Country', items: ['France'] }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Using local variable, the JSON data can also be bound to the pivot table using [DataManager](#) option with the help of [JsonAdaptor](#). Here the instance of [DataManager](#) holding JSON data is assigned to [dataSource](#) property under [dataSourceSettings](#). The use of [DataManager](#) is optional here.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { DataManager, JsonAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let pivotObj;
  let dataSource = new DataManager({
    json: pivotData,
    adaptor: new JsonAdaptor
  });
  let dataSourceSettings = {
    dataSource: dataSource,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France'] }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import { DataManager, JsonAdaptor, Query, ReturnOption } from
 '@syncfusion/ej2-data';

```

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let pivotObj: PivotViewComponent;
  let dataSource: DataManager = new DataManager({
    json: pivotData,
    adaptor: new JsonAdaptor
  });
  let dataSourceSettings: IDataOptions = {
    dataSource: dataSource,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

In the meantime, the JSON data from the local *.json file type can also be connected to the pivot table via the file uploader option. Here, the resulting string after uploading the file needs to be converted to JSON data that can be assigned to the [dataSource](#) property under [dataSourceSettings](#). The following code example illustrates the same.

`javascript

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Uploader } from '@syncfusion/ej2-inputs';
function App() {
  // Step 1: Initiate the file uploader
  let uploadObj: Uploader = new Uploader({
  });
  uploadObj.appendTo('#fileupload');
  let input = document.querySelector('input[type="file"]');
  // Step 2: Add the event listener which fires when the *.JSON file is uploaded.
  input.addEventListener('change', function (e: Event) {
  // Step 3: Initiate the file reader

```

```

let reader: FileReader = new FileReader();
reader.onload = function () {
// Step 4: Getting the string output which is to be parsed as JSON.
let result: string[][] = JSON.parse(reader.result as string);
let dataSourceSettings: IDataOptions = {
// Step 5: The JSON result to be bound as data source.
dataSource: result
// Step 6: The appropriate report needs to be provided here.
}
reader.readAsText((input as any).files[0]);
}
});
let pivotObj: PivotViewComponent;
return <PivotViewComponent ref={d => pivotObj = d} id = 'PivotView' height = { 350} dataSourceSettings
= { dataSourceSettings } ></PivotViewComponent>
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Binding JSON data via remote

In-order to bind remote JSON data, mention the endpoint [URL](#) under [dataSourceSettings](#) property. The [URL](#) property supports both direct downloadable file (*.json) and web service URL.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let dataSourceSettings = {
    url: 'https://cdn.syncfusion.com/data/sales-analysis.json',
    expandAll: false,
    rows: [
      { name: 'EnerType', caption: 'Energy Type' },
    ],
    columns: [
      { name: 'EneSource', caption: 'Energy Source' }
    ],
    values: [
      { name: 'PowUnits', caption: 'Units (GWh)' },
      { name: 'ProCost', caption: 'Cost (MM)' }
    ],
    filters: []
  };
};

```



```

    let pivotObj;
    return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let dataSourceSettings: IDataOptions = {
        url: 'https://cdn.syncfusion.com/data/sales-analysis.json',
        expandAll: false,
        rows: [
            { name: 'EnerType', caption: 'Energy Type' },
        ],
        columns: [
            { name: 'EneSource', caption: 'Energy Source' }
        ],
        values: [
            { name: 'PowUnits', caption: 'Units (GWh)' },
            { name: 'ProCost', caption: 'Cost (MM)' }
        ],
        filters: []
    };
    let pivotObj: PivotViewComponent;
    return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

CSV

For CSV data binding, the `type` property under [dataSourceSettings](#) needs to be set as `CSV` mandatorily.

The CSV format is considered to be the most compact format compared to JSON since it is half the size of JSON. This helps to reduce the bandwidth while transferring to the browser.

Binding CSV data via local

In-order to bind local CSV data to the pivot table, user needs to convert it as string array and then directly assign it to the [dataSource](#) property under [dataSourceSettings](#).

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { csvdata } from './datasource';
function App() {

```

```

    let dataSourceSettings = {
      dataSource: getCSVData(),
      type: 'CSV',
      expandAll: false,
      columns: [{ name: 'Item Type' }, { name: 'Sales Channel' }],
      filters: [],
      drilledMembers: [{ name: 'Item Type', items: ['Baby Food' ]}],
      formatSettings: [{ name: 'Total Cost', format: 'C0' }, { name:
'Total Revenue', format: 'C0' }, { name: 'Total Profit', format: 'C0' }],
      rows: [{ name: 'Region' }, { name: 'Country' }],
      values: [{ name: 'Total Cost' }, { name: 'Total Revenue' }, { name:
'Total Profit' }],
    };
    function getCSVData() {
      const dataSource = [];
      const jsonObject = csvdata.split(/\r?\n|\r/);
      for (let i = 0; i < jsonObject.length; i++) {
        if (!isNullOrUndefined(jsonObject[i]) && jsonObject[i] !== '') {
          dataSource.push(jsonObject[i].split(','));
        }
      }
      return dataSource;
    }
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
  }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { csvdata } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: getCSVData(),
    type: 'CSV',
    expandAll: false,
    columns: [{ name: 'Item Type' }, { name: 'Sales Channel' } ],
    filters: [],
    drilledMembers: [{ name: 'Item Type', items: ['Baby Food' ]}],
    formatSettings: [{ name: 'Total Cost', format: 'C0' }, { name: 'Total
Revenue', format: 'C0' }, { name: 'Total Profit', format: 'C0' }],
    rows: [{ name: 'Region' }, { name: 'Country' }],
    values: [{ name: 'Total Cost' }, { name: 'Total Revenue' }, { name:
'Total Profit' }],
  };
  function getCSVData() {
    const dataSource = [];
    const jsonObject = csvdata.split(/\r?\n|\r/);

```

```

    for (let i = 0; i < jsonObject.length; i++) {
        if (!isNullOrUndefined(jsonObject[i]) && jsonObject[i] !== '') {
            dataSource.push(jsonObject[i].split(','));
        }
    }
    return dataSource;
}
let pivotObj: PivotViewComponent;
return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

In the meantime, the CSV data from the local *.csv file type can also be connected to the pivot table via the file uploader option. Here, the resulting string after uploading the file needs to be converted to string array that can be assigned to the [dataSource](#) property under [dataSourceSettings](#). The following code example illustrates the same.

`javascript

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Uploader } from '@syncfusion/ej2-inputs';

function App() {
    // Step 1: Initiate the file uploader
    let uploadObj: Uploader = new Uploader({
    });
    uploadObj.appendTo('#fileupload');
    let input = document.querySelector('input[type="file"]');
    // Step 2: Add the event listener which fires when the *.CSV file is uploaded.
    input.addEventListener('change', function (e: Event) {
        // Step 3: Initiate the file reader
        let reader: FileReader = new FileReader();
        reader.onload = function () {
            // Step 4: Getting the string output which is to be converted as string[[]].
            let result: string[][] = (reader.result as string).split('\n').map(function (line) {
                return line.split(',');
            });
            let dataSourceSettings: IDataOptions = {

```

```
// Step 5: The string[][] result to be bound as data source
dataSource: result,
type: 'CSV',
// Step 6: The appropriate report needs to be provided here.
}
reader.readAsText((input as any).files[0]);
}
});

let pivotObj: PivotViewComponent;

return (<PivotViewComponent ref={d => pivotObj = d} id = 'PivotView' height = { 350}
dataSourceSettings = { dataSourceSettings } > </PivotViewComponent>);
}

export default App;

ReactDOM.render(<App />, document.getElementById('sample'));
`
```

Binding CSV data via remote

In-order to bind remote CSV data, mention the endpoint [URL](#) under [dataSourceSettings](#) property. The [URL](#) property supports both direct downloadable file (*.csv) and web service URL.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let dataSourceSettings = {
    url: 'https://bi.syncfusion.com/productservice/api/sales',
    type: 'CSV',
    expandAll: false,
    enableSorting: true,
    formatSettings: [{ name: 'Total Cost', format: 'C0' }, { name:
'Total Revenue', format: 'C0' }, { name: 'Total Profit', format: 'C0' }],
    drilledMembers: [{ name: 'Item Type', items: ['Baby Food' ]}],
    rows: [
      { name: 'Region' },
      { name: 'Country' }
    ],
    columns: [
      { name: 'Item Type' },
      { name: 'Sales Channel' }
    ],
    values: [
      { name: 'Total Cost' },
      { name: 'Total Revenue' },
      { name: 'Total Profit' }
    ],
    filters: []
  }
```

```

    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
  }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let dataSourceSettings: IDataOptions = {
    url: 'https://bi.syncfusion.com/productservice/api/sales',
    type: 'CSV',
    expandAll: false,
    enableSorting: true,
    formatSettings: [{ name: 'Total Cost', format: 'C0' }, { name: 'Total Revenue', format: 'C0' }, { name: 'Total Profit', format: 'C0' }],
    drilledMembers: [{ name: 'Item Type', items: ['Baby Food'] }],
    rows: [
      { name: 'Region' },
      { name: 'Country' }
    ],
    columns: [
      { name: 'Item Type' },
      { name: 'Sales Channel' }
    ],
    values: [
      { name: 'Total Cost' },
      { name: 'Total Revenue' },
      { name: 'Total Profit' }
    ],
    filters: []
  };
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Remote Data Binding

To interact with remote data source, provide the endpoint url within [DataManager](#) along with appropriate [adaptor](#). By default, [DataManager](#) uses [ODataAdaptor](#) for remote data-binding.

Binding with OData services

OData is a standardized protocol for creating and consuming data. User can retrieve data from OData service using the [DataManager](#) class. Refer to the following code example for remote data binding using OData service.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { DataManager, ODataAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let dataSource = new DataManager({
        url:
        'https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders/',
        adaptor: new ODataAdaptor,
        crossDomain: true
    });
    let dataSourceSettings = {
        dataSource: dataSource,
        expandAll: true,
        filters: [],
        columns: [{ name: 'OrderDate' }, { name: 'ShipCity' }],
        rows: [{ name: 'OrderID' }, { name: 'CustomerID' }],
        values: [{ name: 'Freight' }]
    };
    let pivotObj;
    return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import { DataManager, ODataAdaptor, Query, ReturnOption } from
 '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let dataSource: DataManager = new DataManager({
        url:
        'https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders/',
        adaptor: new ODataAdaptor,
        crossDomain: true
    });
    let dataSourceSettings: IDataOptions = {
        dataSource: dataSource,
        expandAll: true,
        filters: [],
        columns: [{ name: 'OrderDate' }, { name: 'ShipCity' }],
        rows: [{ name: 'OrderID' }, { name: 'CustomerID' }],
        values: [{ name: 'Freight' }]
    };
    let pivotObj: PivotViewComponent;
    return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>
};
```

```
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Binding with OData V4 services

The OData V4 is an improved version of OData protocols, and the [DataManager](#) class can be used to retrieve and consume OData V4 services. For more details on OData V4 services, refer to the [OData documentation](#). To bind OData V4 service, use the [ODataV4Adaptor](#).

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj;
    let dataSource = new DataManager({
        adaptor: new ODataV4Adaptor,
        url:
        'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/?$top=7',
        crossDomain: true
    });
    let dataSourceSettings = {
        dataSource: dataSource,
        expandAll: true,
        filters: [],
        columns: [{ name: 'OrderDate' }, { name: 'ShipCity' }],
        rows: [{ name: 'OrderID' }, { name: 'CustomerID' }],
        values: [{ name: 'Freight' }]
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import { DataManager, ODataV4Adaptor, Query, ReturnOption } from
 '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj: PivotViewComponent;
    let dataSource: DataManager = new DataManager({
        adaptor: new ODataV4Adaptor,
        url:
        'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/?$top=7',
        crossDomain: true
    });
    let dataSourceSettings: IDataOptions = {
        dataSource: dataSource,
```

```

    expandAll: true,
    filters: [],
    columns: [{ name: 'OrderDate' }, { name: 'ShipCity' }],
    rows: [{ name: 'OrderID' }, { name: 'CustomerID' }],
    values: [{ name: 'Freight' }]
  };
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Web API

User can use [WebApiAdaptor](#) to bind pivot table with Web API created using OData endpoint.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let dataSource = new DataManager({
    url: 'https://bi.syncfusion.com/northwindservice/api/orders',
    adaptor: new WebApiAdaptor,
    crossDomain: true
  });
  let dataSourceSettings = {
    dataSource: dataSource,
    expandAll: true,
    filters: [],
    columns: [{ name: 'ProductName', caption: 'Product Name' }],
    rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
    formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
    values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }]
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import { DataManager, WebApiAdaptor, Query, ReturnOption } from
 '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let dataSource: DataManager = new DataManager({

```



```

url: 'https://bi.syncfusion.com/northwindservice/api/orders',
adaptor: new WebApiAdaptor,
crossDomain: true
});
let dataSourceSettings: IDataOptions = {
  dataSource: dataSource,
  expandAll: true,
  filters: [],
  columns: [{ name: 'ProductName', caption: 'Product Name' }],
  rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
  formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
  values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }]
};
let pivotObj: PivotViewComponent;
return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Querying in Data Manager

By default, the data manager retrieves all the data from the provider which is mapped in it. The data from the provider can be filtered, sorted, paged, etc. by setting the own query in `defaultQuery` property in the data manager instance.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { DataManager, ODataAdaptor, Query } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let pivotObj;
  let dataSource = new DataManager({
    url:
'https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders',
    adaptor: new ODataAdaptor(),
    crossDomain: true
  }, new Query().take(2));
  let dataSourceSettings = {
    dataSource: dataSource,
    expandAll: false,
    columns: [{ name: 'CustomerID', caption: 'Customer ID' }],
    rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
    values: [{ name: 'Freight' }]
  };
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { DataManager, ODataAdaptor, Query, ReturnOption } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj: PivotViewComponent;
    let dataSource: DataManager = new DataManager(
        {
            url:
                'https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders',
            adaptor: new ODataAdaptor(),
            crossDomain: true
        },
        new Query().take(2)
    );
    let dataSourceSettings: IDataOptions = {
        dataSource: dataSource,
        expandAll: false,
        columns: [{ name: 'CustomerID', caption: 'Customer ID' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name: 'ShipCity', caption: 'Ship City' }],
        values: [{ name: 'Freight' }]
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Mapping

One can define field information like alias name (caption), data type, aggregation type, show and hide subtotals etc. using the [fieldMapping](#) property under [dataSourceSettings](#). The available options are,

- [name](#) - It is to specify the appropriate field name.
- [caption](#) - It is to set the alias name (caption) to the specific field. Instead of actual field name, the alias name (caption) will be set in the UI of the pivot table.
- [type](#) - It is to display values in the pivot table with appropriate aggregation such as sum, product, count, average, minimum, maximum, etc. Its default value is **sum**. This option is applicable only for relational data source.
- [axis](#) - It will help to display the field in specified axis such as row/column/value/filter axis of the pivot table.
- [showNoDataItems](#) - It is to show all the members of a specific field to the pivot table, even if there are no data in the intersection of the row and column. The default value is **false**. This option is applicable only for relational data source.
- [baseField](#) - For the aggregate types like "DifferenceFrom" or "PercentageOfDifferenceFrom" or "PercentageOfParentTotal", selective field is assigned for comparison via this property.

- [baseItem](#) For the aggregate types like "DifferenceFrom" or "PercentageOfDifferenceFrom" or "PercentageOfParentTotal", selective member in a field is assigned for comparison via this property.
- [expandAll](#) - It is to expand or collapse all headers of a specific field in row and column axes of the pivot table. The default value is **false**.
- [showSubTotals](#) - It is to show or hide sub-totals of a specific field in row and column axis of the pivot table. The default value is **true**.
- [isNamedSet](#) - It is to set whether the specified field is named set or not. In general, the named set is a set of dimension members or a set expression (MDX query) to be created as a dimension in the SSAS OLAP cube itself. The default value is **false** and this option is applicable only for OLAP data source.
- [isCalculatedField](#) - It is to set whether the specified field is a calculated field or not. In general, a calculated field is created from the bound data source or using simple formula with basic arithmetic operators in the pivot table. The default value is **false** and this option is applicable only for OLAP data source.
- [showFilterIcon](#) - It is to show or hide the filter icon of a specific field which will be displayed on the button of the grouping bar and field list UI. This filter icon is used to filter the members of a specified field at runtime in the pivot table. The default value is **true**.
- [showSortIcon](#) - It is to show or hide the sort icon of a specific field which will be displayed on the button of the grouping bar and field list UI. This sort icon is used to order members of a specified field either in ascending or descending at runtime. The default value is **true**.
- [showRemoveIcon](#) - It is to show or hide the remove icon of a specific field which will be displayed on the button of the grouping bar and field list UI. This remove icon is used to remove the specified field during runtime. The default value is **true**.
- [showValueTypeIcon](#) - It is to show or hide the value type icon of a specific field which will be displayed on the button of the grouping bar and field list UI. This value type icon helps to select the appropriate aggregation type to specified value field at runtime. The default value is **true**.
- [showEditIcon](#) - It is to show or hide the edit icon of a specific field which will be displayed on the button of the grouping bar and field list UI. This edit icon is used to modify caption, formula, and format of a specified calculated field at runtime. The default value is **true**.
- [allowDragAndDrop](#) - It is to restrict specific field's button from being dragged on runtime in the grouping bar and field list UI. This will prevent from altering the current report. The default value is **true**.
- [dataType](#) - It is to specify the type of the field like 'string', 'number', 'datetime', 'date', and 'boolean'.
- [groupName](#) - It is to display fields in the field list UI by grouping them under the desired folder name.

The main purpose of these mapping options is to configure each field that is not part of the initial pivot report. Even if any field that is part of this mapping is defined here, the value set in the initial report will have the highest preceding.

This option is applicable only for relational data source.

In the below code sample, visibility of the field button icons are configured.

INDEX.JSX

```
import { GroupingBar, FieldList, Inject, PivotViewComponent } from
'@syncfusion/ej2-react-pivotview';
```

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }],
    dataSource: pivotData,
    expandAll: false,
    allowLabelFilter: true,
    allowValueFilter: true,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }],
    fieldMapping: [
      { name: 'Quarter', showSortIcon: false },
      { name: 'Products', showFilterIcon: false, showRemoveIcon: false
    },
      { name: 'Amount', showValueTypeIcon: false, caption: 'Sold
Amount' } ],
  ];
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showGroupingBar={true}
showFieldList={true}><Inject services={[GroupingBar,
FieldList]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { GroupingBar, FieldList, IDataOptions, IDataset, Inject,
PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    allowLabelFilter: true,
    allowValueFilter: true,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }],
    fieldMapping: [
      { name: 'Quarter', showSortIcon: false },
      { name: 'Products', showFilterIcon: false, showRemoveIcon: false
    },
      { name: 'Amount', showValueTypeIcon: false, caption: 'Sold
Amount' } ],
  };

```

```

    ]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showGroupingBar={true}
showFieldList={true} ><Inject services={[GroupingBar,
FieldList]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Values in row axis

By default, the value fields are plotted in column axis. To plot those fields in row axis, use the [valueAxis](#) property by setting its value as **row**. By default, it holds the value **column**.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    valueAxis: 'row'
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,

```

```

    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    valueAxis: 'row'
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Values at different positions

By default, the value fields are placed at the end of the row or column axis. To place those value fields in different positions, use the [valueIndex](#) property and set the value to an appropriate index position. Its default value is -1, which denotes the last position. The [valueIndex](#) property is dependent on the [valueAxis](#) property.

This support is only available for relational data sources. Also, enable the [showValuesButton](#) property in the grouping bar and field list UI to **true** to re-arrange the values fields at different positions via user interaction.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: true,
    filters: [],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    valueIndex: 1
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';

```

```
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: true,
    filters: [],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    valueIndex: 1
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Show 'no data' items

By default, the pivot table only shows the field item if it has data in its row or column combination. To show all items that do not have data in row and column combination in the pivot table, use the [showNoDataItems](#) property by settings its value to **true** for the desired fields. In the following code sample, rows of the "County" and "State" fields do not have data in all combination with "Date" column field.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { noData } from './datasource';
function App() {
  let dataSourceSettings = {
    dataSource: noData,
    expandAll: true,
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    columns: [{ name: 'Date', showNoDataItems: true }],
    values: [{ name: 'Quantity', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
    rows: [{ name: 'Country', showNoDataItems: true }, { name: 'State',
showNoDataItems: true }],
    filters: []
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { noData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: noData,
    expandAll: true,
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    columns: [{ name: 'Date', showNoDataItems: true }],
    values: [{ name: 'Quantity', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country', showNoDataItems: true }, { name: 'State',
showNoDataItems: true }],
    filters: []
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Show value headers always

To show value header always in pivot table, even if it holds a single value, use the [alwaysShowValueHeader](#) property by settings its value as **true**.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }],
    alwaysShowValueHeader: true
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```


INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }],
    alwaysShowValueHeader: true
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Customize empty value cells

User can show custom string in empty value cells using the [emptyCellsTextContent](#) property in [dataSourceSettings](#). Since the property is of string data type, user can fill empty value cells with any value like "0", "-", "*", "(blank)", etc. Its common for all value fields and can be configured through code behind.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    emptyCellsTextContent: '*',
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C2' }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
```

```

    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
  }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    emptyCellsTextContent: '*',
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C2' }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Event

Load

The event [load](#) fires before initiate rendering of pivot table. It holds following parameters like [dataSourceSettings](#), [fieldsType](#) and [PivotView](#). In this event user can customize data source settings before initiating pivot table render module.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],

```

```

        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C2' }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    };
    let pivotObj;
    function onLoad(args) {
        args.dataSourceSettings.emptyCellsTextContent = "###";
        args.dataSourceSettings.columns[0].caption = "Full Year";
        args.dataSourceSettings.expandAll = true;
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} load={onLoad.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, LoadEventArgs } from
'@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C2' }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    }
    let pivotObj: PivotViewComponent;
    function onLoad(args: LoadEventArgs): void {
        args.dataSourceSettings.emptyCellsTextContent = "###";
        args.dataSourceSettings.columns[0].caption = "Full Year";
        args.dataSourceSettings.expandAll = true;
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} load={onLoad.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}

```

```
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

EnginePopulated

The event [enginePopulated](#) is triggered after engine is populated. It has following parameters - `dataSourceSettings`, `pivotFieldList` and `pivotValues`.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C2' }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
  };
  let pivotObj;
  function enginePopulated(args) {
    // triggers after engine gets populated
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enginePopulated={enginePopulated.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent,
EnginePopulatedEventArgs } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
  };
  let pivotObj;
  function enginePopulated(args: EnginePopulatedEventArgs) {
    // triggers after engine gets populated
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enginePopulated={enginePopulated.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

```

        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C2' }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    }
    let pivotObj: PivotViewComponent;
    function enginePopulated(args: EnginePopulatedEventArgs): void {
        // triggers after engine gets populated
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enginePopulated={enginePopulated.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

EnginePopulating

The event [enginePopulating](#) triggers before the pivot engine starts to populate and allows to customize the pivot datasource settings. It has following parameter `dataSourceSettings`.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C2' }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    };
    let pivotObj;
    function enginePopulating(args) {
        args.dataSourceSettings.columns[0].caption = "Full Year";
        args.dataSourceSettings.expandAll = true;
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enginePopulating={enginePopulating.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent,
EnginePopulatingEventArgs } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C2' }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
  }
  let pivotObj: PivotViewComponent;
  function enginePopulating(args: EnginePopulatingEventArgs): void {
    args.dataSourceSettings.columns[0].caption = "Full Year";
    args.dataSourceSettings.expandAll = true;
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enginePopulating={enginePopulating.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

See Also

- [Aggregation](#)
- [Show/Hide Totals](#)
- [Customize number, date, and time values](#)
- [Server Side Engine \(Optional\)](#)

Connecting to data source

MySQL in EJ2 React Pivotview Component

This section describes how to retrieve data from a MySQL database using [MySqlClient](#) and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch MySQL data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).

Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name

MyWebService

Location

C:\Users\username\source\repos

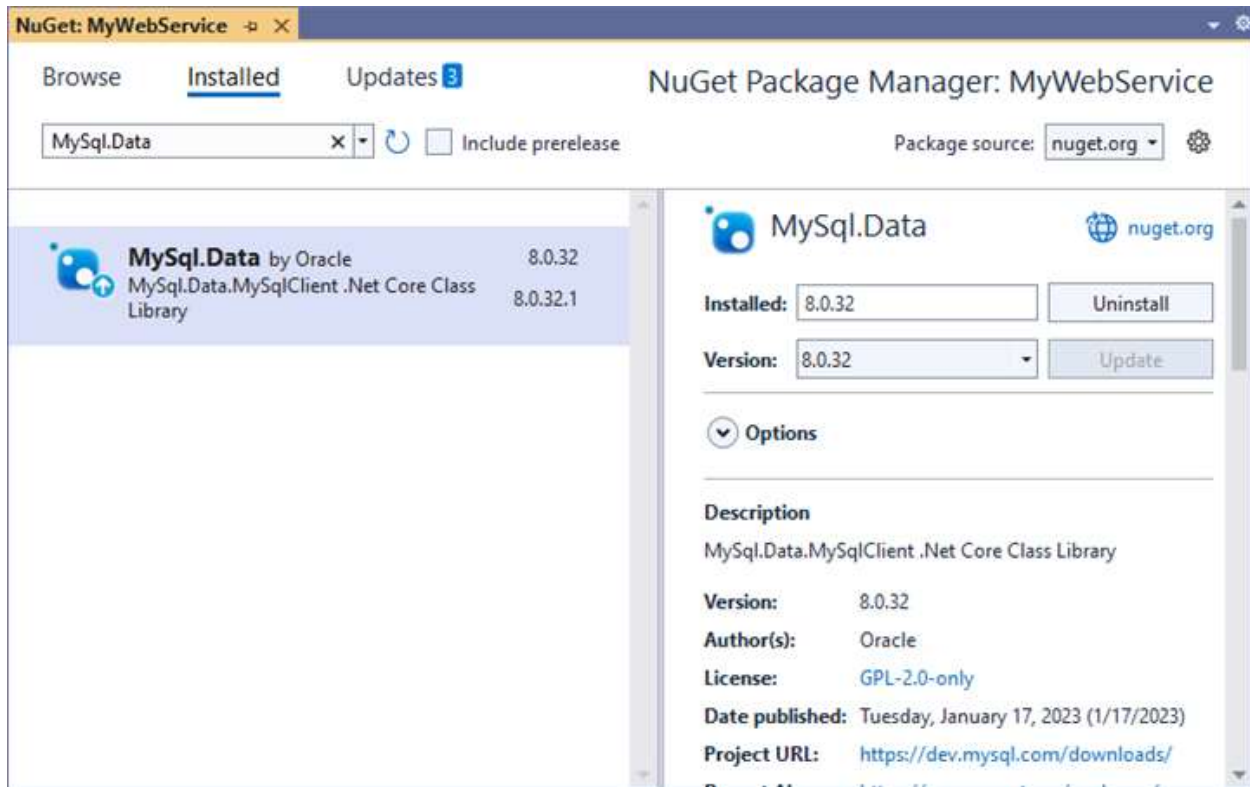
Solution name ⓘ

MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a MySQL Server using the **MySqlClient** in our application, we need to install the [MySql.Data](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **MySql.Data** and install it.



3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **MySQLConnection** helps to connect the MySQL database. Next, using **MySQLCommand** and **MySQLDataAdapter** you can process the desired query string and retrieve data from the MySQL database. The **Fill** method of the **MySQLDataAdapter** is used to populate the retrieved data into a **DataTable** as shown in the following code snippet.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using MySQL.Data.MySqlClient;
using Newtonsoft.Json;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        public dynamic GetMySQLResult()
        {
```



```
// Replace with your own connection string.
MySQLConnection connection = new MySqlConnection("<Enter your valid connection string here>");
connection.Open();
MySQLCommand command = new MySQLCommand("SELECT * FROM orders", connection);
MySQLDataAdapter dataAdapter = new MySQLDataAdapter(command);
DataTable dataTable = new DataTable();
dataAdapter.Fill(dataTable);
connection.Close();
return dataTable;
}
}
}
```

5. In the **Get()** method of the **PivotController.cs** file, the **GetMySQLResult** method is used to retrieve the MySQL data as a **DataTable**, which is then serialized into JSON string using **JsonConvert.SerializeObject()**.

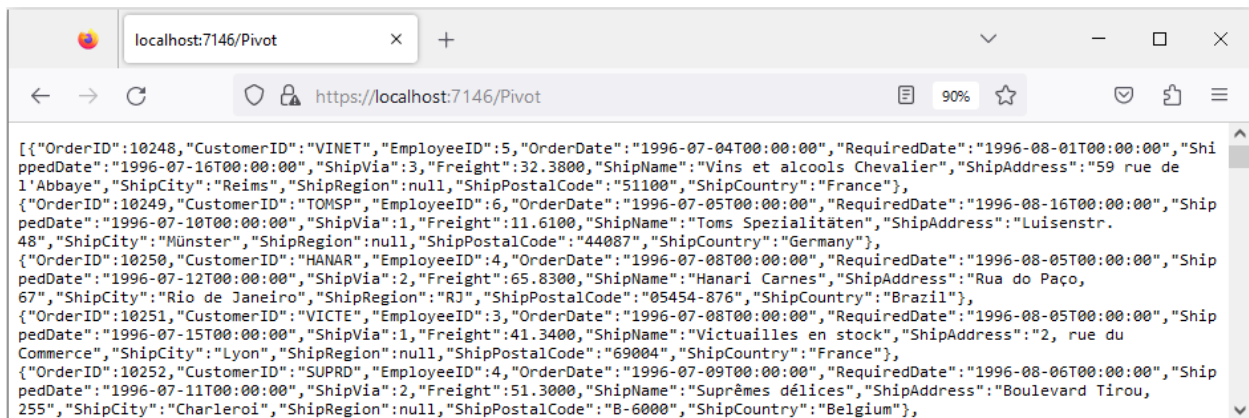
```
`csharp
using Microsoft.AspNetCore.Mvc;
using MySql.Data.MySqlClient;
using Newtonsoft.Json;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetMySQLResult")]
        public object Get()
        {
            return JsonConvert.SerializeObject(GetMySQLResult());
        }
        public dynamic GetMySQLResult()
        {

```

```
// Replace with your own connection string.
MySQLConnection connection = new MySqlConnection("<Enter your valid connection string here>");
connection.Open();
MySQLCommand command = new MySqlCommand("SELECT * FROM orders", connection);
MySQLDataAdapter dataAdapter = new MySQLDataAdapter(command);
DataTable dataTable = new DataTable();
dataAdapter.Fill(dataTable);
connection.Close();
return dataTable;
}
}
}
```

6. Run the application and it will be hosted within the URL <https://localhost:7146>.

7. Finally, the retrieved data from MySQL database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:7146/Pivot>, as shown in the browser page below.



Connecting the Pivot Table to a MySQL database using the Web API service

1. Create a simple React Pivot Table by following the **"Getting Started"** documentation [link](#).
2. Map the hosted Web API's URL link <https://localhost:7146/Pivot> to the Pivot Table in **app.ts** by using the **url** property under [dataSourceSettings](#).

```
`typescript
```

```
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
```

```

let dataSourceSettings = {
url: 'https://localhost:7146/pivot',
//Other codes here...
};

return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}>
<Inject services={[FieldList]}/></PivotViewComponent>);
};

export default App;
`

```

3. Frame and set the report based on the data retrieved from the MySQL database.

```

`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
let dataSourceSettings = {
url: 'https://localhost:7146/pivot',
enableSorting: true,
expandAll: false,
columns: [{ name: 'ShipName' }],
values: [{ name: 'Freight', caption: 'Sum of Freight' }],
rows: [{ name: 'ShipCity' }],
filters: []
};

return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}>
<Inject services={[FieldList]}/></PivotViewComponent>);
};

export default App;
`

```

When you run the sample, the resulting pivot table will look like this:



	Alfred's Futterkist	Alfreds Futterkist	Ana Trujillo Empa	Antonio Moreno	Around the Horn	B's	
Aachen							
Albuquerque							
Anchorage							
Barcelona							
Barquisimeto							
Bergamo							
Berlin	196.12000000...	29.46					
Bern							

Explore our React Pivot Table sample and ASP.NET Core Web Application to extract data from a MySQL database and bind to the Pivot Table in [this](#) GitHub repository.

Microsoft SQL Server in EJ2 React Pivotview Component

This section describes how to retrieve data from SQL Server database using [Microsoft SqlClient](#) and bind it to the Pivot Table via a Web API controller.

Steps to connect the SQL Server database via Web API application

1. Download the ASP.NET Core Web Application from [this](#) GitHub repository.
2. The application named as **PivotController** (server-side) that is downloaded from the above GitHub repository includes the following files.

- **PivotController.cs** file under **Controllers** folder – This helps to do data communication with Pivot Table.
- **Database1.mdf** file under **App_Data** folder – This MDF (Master Database File) file contains example data.

3. In the Web API controller (aka, PivotController), **SqlConnection** helps to connect the SQL database (that is, Database1.mdf). Next, using **SqlCommand** and **SqlDataAdapter** you can process the desired SQL query string and retrieve data from the database. The **Fill** method of the DataAdapter is used to populate the SQL data into a **DataTable** as shown in the following code snippet.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using System.Data;
using System.Data.SqlClient;
namespace PivotController.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
```

```

{
private static DataTable FetchSQLResult()
{
string conSTR = @"<Enter your valid connection string here>";
string xquery = "SELECT * FROM table1";
SqlConnection sqlConnection = new(conSTR);
sqlConnection.Open();
SqlCommand cmd = new(xquery, sqlConnection);
SqlDataAdapter dataAdapter = new(cmd);
DataTable dataTable = new();
dataAdapter.Fill(dataTable);
return dataTable;
}
}
}
`

```

4. In the **Get()** method of the **PivotController.cs** file, the **FetchSQLResult** method is used to retrieve the SQL data as a **DataTable**, which is then serialized into JSON using **JsonConvert.SerializeObject()**.

```

`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System.Data;
using System.Data.SqlClient;
namespace PivotController.Controllers
{
[ApiController]
[Route("[controller]")]
public class PivotController : ControllerBase
{
[HttpGet(Name = "GetSQLResult")]
public object Get()
{
return JsonConvert.SerializeObject(FetchSQLResult());
}
}
}

```

```

}

private static DataTable FetchSQLResult()
{
    string conSTR = @"<Enter your valid connection string here>";
    string xquery = "SELECT * FROM table1";
    SqlConnection sqlConnection = new(conSTR);
    sqlConnection.Open();
    SqlCommand cmd = new(xquery, sqlConnection);
    SqlDataAdapter dataAdapter = new(cmd);
    DataTable dataTable = new();
    dataAdapter.Fill(dataTable);
    return dataTable;
}
}
,

```

5. Run the web application (aka, PivotController) and it will be hosted within the URL <https://localhost:7139>.

6. Finally, the retrieved data from SQL Server which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:7139/pivot>, as shown in the browser page below.



```

[{"Product": "Bike", "State": "New South Wales", "Redueby": 10.00, "Date": "FY 2005", "Quantity": 3, "Amount": 500.000, "Category": "Small Vehicle", "Country": "Australia"},
{"Product": "Bus", "State": "Sydney", "Redueby": 40.00, "Date": "FY 2006", "Quantity": 4, "Amount": 6000.000, "Category": "Large Vehicle", "Country": "Australia"},
{"Product": "Truck", "State": "Kingston", "Redueby": 33.10, "Date": "FY 2007", "Quantity": 67, "Amount": 5000.000, "Category": "Large Vehicle", "Country": "Jamaica"},
{"Product": "Cars", "State": "St Ann", "Redueby": 5555.00, "Date": "FY 2009", "Quantity": 33, "Amount": 70030.000, "Category": "Large Vehicle", "Country": "Jamaica"},
{"Product": "Bike", "State": "Ontario", "Redueby": 200.00, "Date": "FY 2010", "Quantity": 45, "Amount": 60000.000, "Category": "Small Vehicle", "Country": "Canada"},
{"Product": "Cars", "State": "Montreal", "Redueby": 110.00, "Date": "FY 2009", "Quantity": 12, "Amount": 70450.000, "Category": "Small Vehicle", "Country": "Canada"},
{"Product": "Bike2", "State": "Port of Spain", "Redueby": 0.00, "Date": "FY2009", "Quantity": 41, "Amount": 56630.000, "Category": "Small Vehicle", "Country": "Trinidad"},
{"Product": "Bike2", "State": "Port of Spain", "Redueby": 0.00, "Date": "FY2009", "Quantity": 41, "Amount": 56630.000, "Category": "Small Vehicle", "Country": "Trinidad"},
{"Product": "Cars", "State": "Montreal", "Redueby": 110.00, "Date": "FY 2009", "Quantity": 12, "Amount": 70450.000, "Category": "Small Vehicle", "Country": "Canada"},
{"Product": "Bike4", "State": "New South Wales", "Redueby": 10.00, "Date": "FY 2005", "Quantity": 3, "Amount": 500.000, "Category": "Small Vehicle", "Country": "Australia"}]

```

Connecting the Pivot Table to the hosted Web API URL

1. Download the react Pivot Table sample from [this](#) GitHub repository.

2. Next, map the hosted Web API's URL link <https://localhost:7139/pivot> to the Pivot Table component in **app.ts** by using the [url](#) property under [dataSourceSettings](#).

```
`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings = {
    url: 'https://localhost:7139/pivot',
    //Other codes here...
  };
  return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
    showFieldList={true}>
    <Inject services={[FieldList]}/></PivotViewComponent>);
};
export default App;
`
```

3. Frame and set the report based on the data retrieved from the SQL database.

```
`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings = {
    url: 'https://localhost:7139/pivot',
    enableSorting: true,
    expandAll: false,
    columns: [{ name: 'Product' }],
    values: [{ name: 'Quantity' }, { name: 'Amount', caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'State' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  };
  return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
    showFieldList={true}>
    <Inject services={[FieldList]}/></PivotViewComponent>);
`
```

```
};
export default App;
`
```

4. Run the sample to get the following result.

	Large Vehicle		Small Vehicle		Grand Total	
	Quantity	Amount	Quantity	Amount	Quantity	Am
▶ Australia	16	24000	12	2000	28	
▶ Canada			420	1649000	420	
▶ Jamaica	400	300120			400	
▶ Trinidad			328	453040	328	
Grand Total	416	324120	760	2104040	1176	

The sample for connecting the Pivot Table to a SQL Server database via an ASP.NET Web application can be found in [this](#) GitHub repository.

PostgreSQL in EJ2 React Pivotview Component

This section describes how to consume data from PostgreSQL database using [Microsoft Npgsql](#) and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch PostgreSQL data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).

Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name
MyWebService

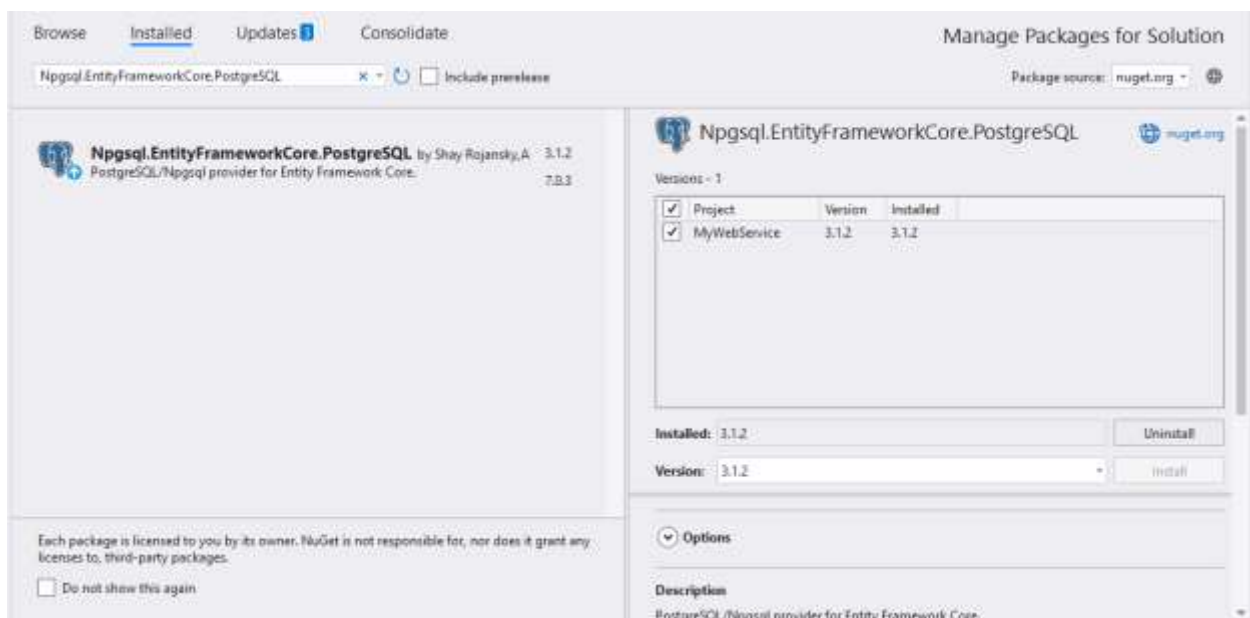
Location
C:\Users\username\source\repos

Solution name
MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a PostgreSQL Server using the **Npgsql** in our application, we need to install the [Npgsql.EntityFrameworkCore.PostgreSQL](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **Npgsql.EntityFrameworkCore.PostgreSQL** and install it.



3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **NpgsqlConnection** helps to connect the PostgreSQL database. Next, using **NpgsqlCommand** and **NpgsqlDataAdapter** you can process the desired PostgreSQL query string and retrieve data from the database. The **Fill** method of the **NpgsqlDataAdapter** is used to populate the retrieved data into a **DataTable** as shown in the following code snippet.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System.Data;
using Npgsql;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        public dynamic GetPostgreSQLResult()
        {
            // Replace with your own connection string.
            NpgsqlConnection connection = new NpgsqlConnection("<Enter your valid connection string here>");
            connection.Open();
            NpgsqlCommand cmd = new NpgsqlCommand("SELECT * FROM tablename", connection);
            NpgsqlDataAdapter da = new NpgsqlDataAdapter(cmd);
            DataTable dt = new DataTable();
            da.Fill(dt);
            connection.Close();
            return dt;
        }
    }
}
```

5. In the **Get()** method of the **PivotController.cs** file, the **GetPostgreSQLResult** method is used to retrieve the PostgreSQL data as a **DataTable**, which is then serialized into JSON using **JsonConvert.SerializeObject()**.

```
`csharp
```

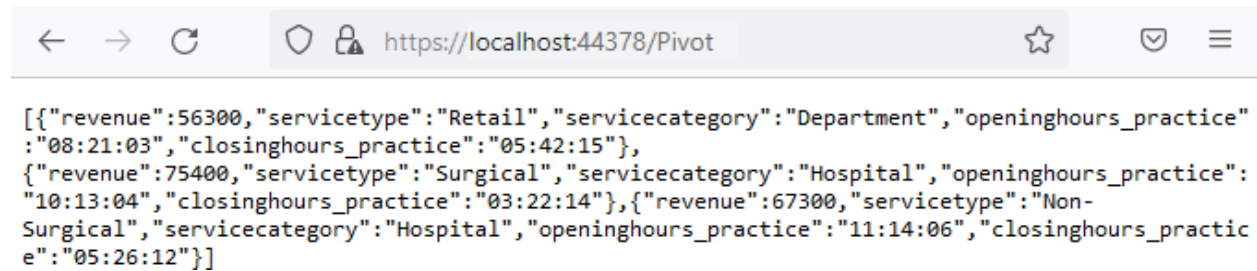
```

using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System.Data;
using Npgsql;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetPostgreSQLResult")]
        public object Get()
        {
            return JsonConvert.SerializeObject(GetPostgreSQLResult());
        }
        public dynamic GetPostgreSQLResult()
        {
            // Replace with your own connection string.
            NpgsqlConnection connection = new NpgsqlConnection("<Enter your valid connection string here>");
            connection.Open();
            NpgsqlCommand cmd = new NpgsqlCommand("SELECT * FROM tablename", connection);
            NpgsqlDataAdapter da = new NpgsqlDataAdapter(cmd);
            DataTable dt = new DataTable();
            da.Fill(dt);
            connection.Close();
            return dt;
        }
    }
}

```

6. Run the application and it will be hosted within the URL <https://localhost:44378/>.

7. Finally, the retrieved data from PostgreSQL database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:44378/Pivot>, as shown in the browser page below.



Connecting the Pivot Table to a PostgreSQL database using the Web API service

1. Create a simple React Pivot Table by following the **"Getting Started"** documentation [link](#).
2. Map the hosted Web API's URL link `https://localhost:44378/Pivot` to the Pivot Table component in **app.ts** by using the `url` property under `dataSourceSettings`.

```
`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings = {
    url: 'https://localhost:44378/pivot',
    //Other codes here...
  };
  return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
    showFieldList={true}>
    <Inject services={[FieldList]}></PivotViewComponent>);
};
export default App;
```

3. Frame and set the report based on the data retrieved from the PostgreSQL database.

```
`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings = {
    url: 'https://localhost:44378/Pivot',
    enableSorting: true,
    columns: [{ name: 'openinghourspractice' }, { name: 'closinghourspractice' }],
```

```

values: [{ name: 'revenue' }],
rows: [{ name: 'servicetype' }, { name: 'servicecategory' }]
};
return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}>
<Inject services={[FieldList]}/></PivotViewComponent>);
};
export default App;
`

```

When you run the sample, the resulting pivot table will look like this:

	▶ 08:21:03	▶ 10:13:04	▶ 11:14:06	Grand Total
▶ Non-Surgical			67300	67300
▶ Retail	56300			56300
▶ Surgical		75400		75400
Grand Total	56300	75400	67300	199000

Explore our React Pivot Table sample and ASP.NET Core Web Application to extract data from a PostgreSQL database and bind to the Pivot Table in [this](#) GitHub repository.

Oracle in EJ2 React Pivotview Component

This section describes how to retrieve data from Oracle database using [Oracle Managed Data Access](#) and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch Oracle data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).

Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name
MyWebService

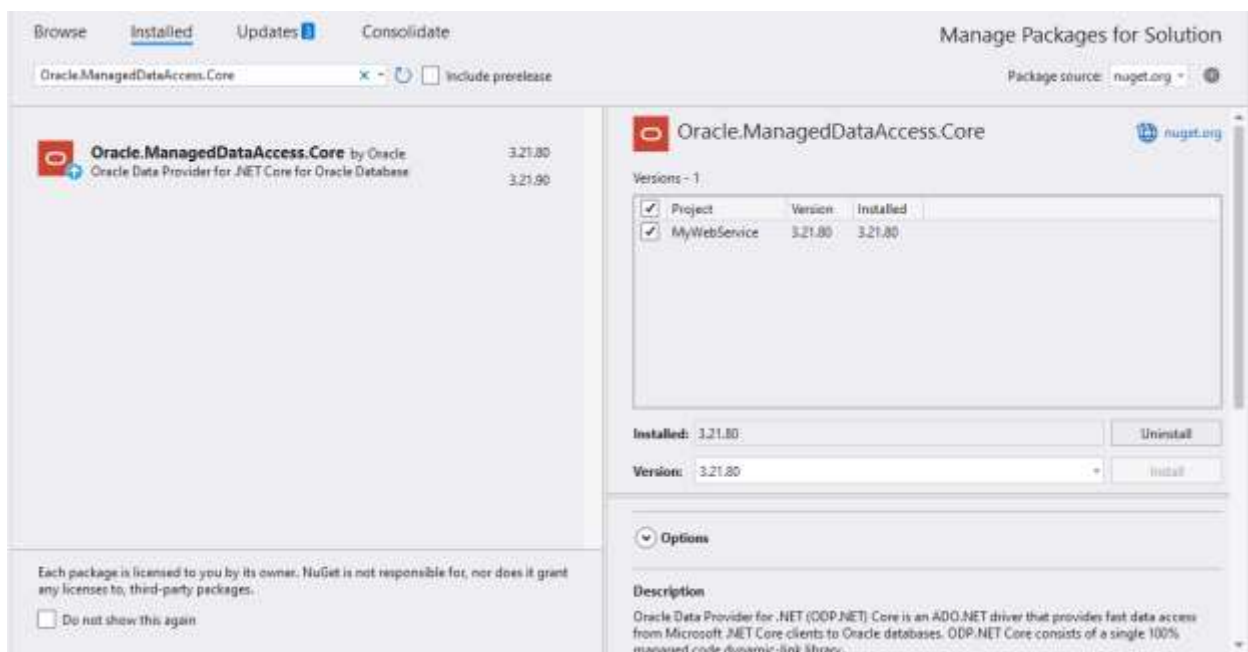
Location
C:\Users\username\source\repos

Solution name
MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a Oracle Server using the **Oracle.ManagedDataAccess.Client** in our application, we need to install the [Oracle.ManagedDataAccess.Core](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **Oracle.ManagedDataAccess.Core** and install it.



3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **OracleConnection** helps to connect the Oracle database. Next, using **OracleCommand** and **OracleDataAdapter** you can process the desired Oracle query string and retrieve data from the database. The **Fill** method of the **OracleDataAdapter** is used to populate the retrieved data into a **DataTable** as shown in the following code snippet.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using Oracle.ManagedDataAccess.Client;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        private static DataTable FetchOracleResult()
        {
            // Replace with your own connection string.
            string connectionString = "<Enter your valid connection string here>";
            OracleConnection oracleConnection = new OracleConnection(connectionString);
            oracleConnection.Open();
            OracleCommand command = new OracleCommand("SELECT * FROM EMPLOYEES", oracleConnection);
            OracleDataAdapter dataAdapter = new OracleDataAdapter(command);
            DataTable dataTable = new DataTable();
            dataAdapter.Fill(dataTable);
            oracleConnection.Close();
            return dataTable;
        }
    }
}
```

5. In the **Get()** method of the **PivotController.cs** file, the **FetchOracleResult()** method is used to retrieve the Oracle data, which is then serialized into JSON using **JsonConvert.SerializeObject()**.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using Oracle.ManagedDataAccess.Client;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetOracleResult")]
        public object Get()
        {
            return JsonConvert.SerializeObject(FetchOracleResult());
        }
        private static DataTable FetchOracleResult()
        {
            // Replace with your own connection string.
            string connectionString = "<Enter your valid connection string here>";
            OracleConnection oracleConnection = new OracleConnection(connectionString);
            oracleConnection.Open();
            OracleCommand command = new OracleCommand("SELECT * FROM EMPLOYEES", oracleConnection);
            OracleDataAdapter dataAdapter = new OracleDataAdapter(command);
            DataTable dataTable = new DataTable();
            dataAdapter.Fill(dataTable);
            oracleConnection.Close();
            return dataTable;
        }
    }
}
```

6. Run the application and it will be hosted within the URL <https://localhost:44346/>.

7. Finally, the retrieved data from Oracle database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:44346/Pivot>, as shown in the browser page below.



Connecting the Pivot Table to a Oracle database using the Web API service

1. Create a simple React Pivot Table by following the “**Getting Started**” documentation [link](#).
2. Map the hosted Web API's URL link <https://localhost:44346/Pivot> to the Pivot Table component in **app.ts** by using the [url](#) property under [dataSourceSettings](#).

```
`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings = {
    url: 'https://localhost:44346/pivot',
    //Other codes here...
  };
  return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
  showFieldList={true}>
    <Inject services={[FieldList]}></PivotViewComponent>);
};
export default App;
```

3. Frame and set the report based on the data retrieved from the Oracle database.

```
`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
```

```
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings = {
    url: 'https://localhost:44346/pivot',
    enableSorting: true,
    expandAll: false,
    columns: [
      { name: 'DEPARTMENT_ID', caption: 'Department ID' },
      { name: 'EMPLOYEE_NAME', caption: 'Employee Name' },
    ],
    rows: [
      { name: 'JOB', caption: 'Job' },
      { name: 'SALARY', caption: 'Salary' }
    ],
    values: [
      { name: 'EMPLOYEE_ID', caption: 'Employee ID' },
      { name: 'CC_EMPLOYEES', caption: 'Employees' },
      { name: 'CCTAXPERCENTAGE', caption: 'Percentage' },
    ],
    filters: []
  };
  return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
    showFieldList={true}>
    <Inject services={[FieldList]}/></PivotViewComponent>);
};
export default App;
```

When you run the sample, the resulting pivot table will look like this:

	▶ 10	▶ 20	▶ 30	Grand Total
▶ ANALYST		15690		15690
▶ CLERK	7934	15245	7900	31079
▶ MANAGER	7782	7566	7698	23046
▶ PRESIDENT	7839			7839
▶ SALESMAN			30518	30518
Grand Total	23555	38501	46116	108172

Explore our React Pivot Table sample and ASP.NET Core Web Application to extract data from a Oracle database and bind to the Pivot Table in [this](#) GitHub repository.

MongoDB in EJ2 React Pivotview Component

This section describes how to consume data from MongoDB database using [MongoDB Driver](#) and [MongoDB Bson](#) libraries and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch MongoDB data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).

Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name
MyWebService

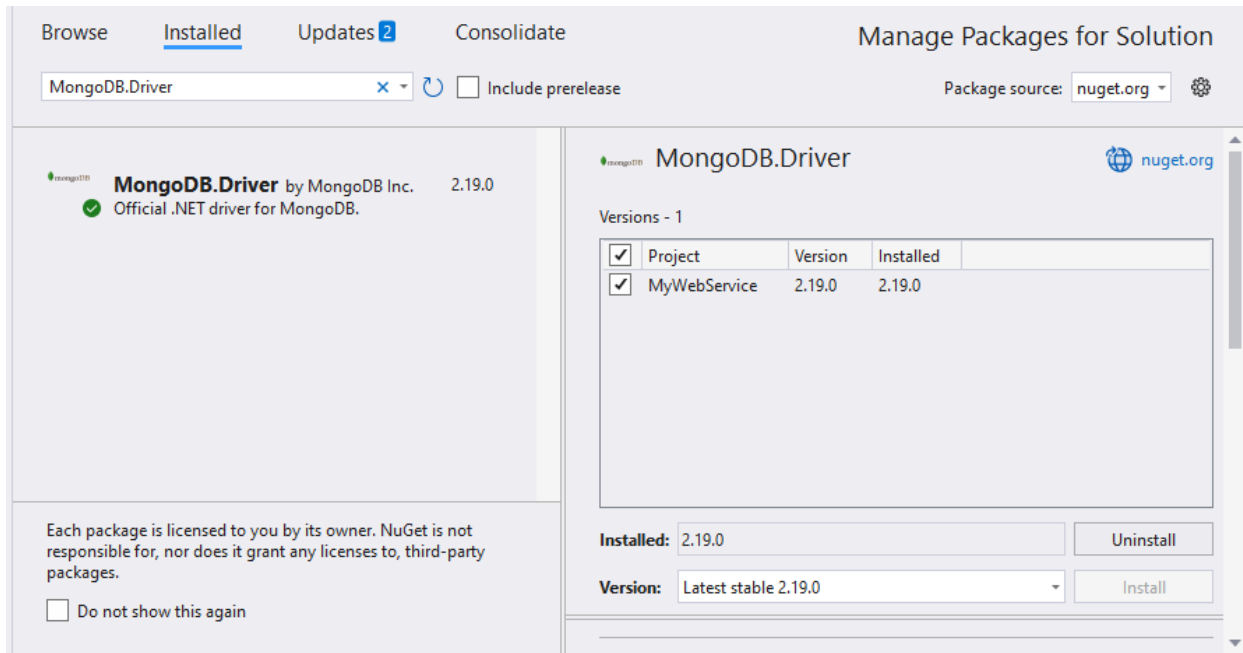
Location
C:\Users\username\source\repos

Solution name ⓘ
MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a MongoDB Server using the **MongoDB.Driver** and **MongoDB.Bson** in our application, we need to install the [MongoDB.Driver](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **MongoDB.Driver** and install it.



3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **MongoClient** helps to connect the MongoDB database. Next, using the **GetDatabase** and **GetCollection** methods, you can retrieve data from the database. The **Find** method of the **IMongoDatabase** is used to populate the retrieved data into a **List**, as shown in the following code snippet.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using MongoDB.Driver;
using MongoDB.Bson;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        private static List<ProductDetails> FetchMongoDbResult()
        {
            // Replace with your own connection string.
            string connectionString = "<Enter your valid connection string here>";
```

```

MongoClient client = new MongoClient(connectionString);
IMongoDatabase database = client.GetDatabase("sample_training");
var collection = database.GetCollection<ProductDetails>("ProductDetails");
return collection.Find(new BsonDocument()).ToList();
}

public class ProductDetails
{
    public ObjectId Id { get; set; }
    public int Sold { get; set; }
    public double Amount { get; set; }
    public string? Country { get; set; }
    public string? Products { get; set; }
    public string? Year { get; set; }
    public string? Quarter { get; set; }
}
}
}
`

```

5. In the **Get()** method of the **PivotController.cs** file, the **FetchMongoDbResult()** method is used to retrieve the MongoDB data as a **List**, which is then serialized into JSON using **JsonConvert.SerializeObject()**.

```

`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using MongoDB.Bson;
using MongoDB.Driver;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetMongoDbResult")]
        public object Get()
    }
}

```

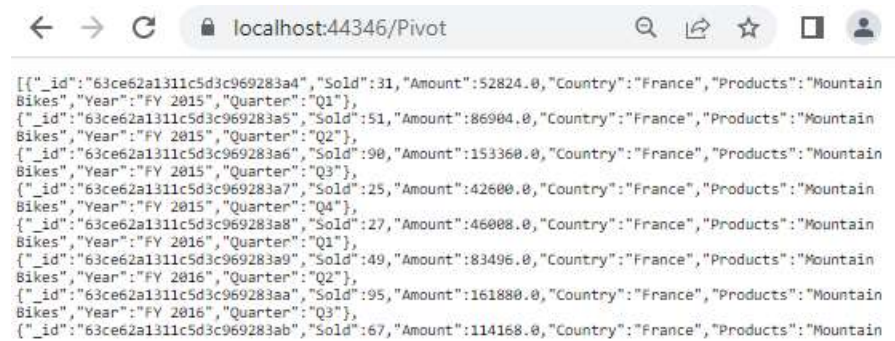
```

{
return JsonConvert.SerializeObject(FetchMongoDbResult());
}
private static List<ProductDetails> FetchMongoDbResult()
{
// Replace with your own connection string.
string connectionString = "<Enter your valid connection string here>";
MongoClient client = new MongoClient(connectionString);
IMongoDatabase database = client.GetDatabase("sample_training");
var collection = database.GetCollection<ProductDetails>("ProductDetails");
return collection.Find(new BsonDocument()).ToList();
}
public class ProductDetails
{
public ObjectId Id { get; set; }
public int Sold { get; set; }
public double Amount { get; set; }
public string? Country { get; set; }
public string? Products { get; set; }
public string? Year { get; set; }
public string? Quarter { get; set; }
}
}
}
`

```

6. Run the web application and it will be hosted within the URL <https://localhost:44346/>.

7. Finally, the retrieved data from MongoDB database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:44346/Pivot>, as shown in the browser page below.



Connecting the Pivot Table to a MongoDB database using the Web API service

1. Create a simple React Pivot Table by following the **"Getting Started"** documentation [link](#).
2. Map the hosted Web API's URL link `https://localhost:44346/Pivot` to the Pivot Table component in `app.ts` by using the `url` property under `dataSourceSettings`.

```
`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings = {
    url: 'https://localhost:44346/pivot',
    //Other codes here...
  };
  return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
    showFieldList={true}>
    <Inject services={[FieldList]}></PivotViewComponent>);
};
export default App;
```

3. Frame and set the report based on the data retrieved from the MongoDB database.

```
`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings = {
    url: 'https://localhost:44346/Pivot',
```

```

enableSorting: true,
columns: [
  { name: 'Year' }
],
values: [
  { name: 'Sold', caption: "Units Sold"},
  { name: 'Amount', caption: "Sold Amount"}
],
rows: [
  { name: 'Country' },
  { name: 'Products' }
]
];

return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}>
<Inject services={{FieldList}}/></PivotViewComponent>);
};

export default App;
`

```

When you run the sample, the resulting pivot table will look like this:

	FY 2015		FY 2016		F
	Units Sold	Sold Amount	Units Sold	Sold Amount	U
> France	729	1160099.5	609	983317	
> Germany	528	845472	667	1067220	
> United Kingdom	782	1263109.5	640	1031630.5	
> United States	682	1085398.5	480	770362	
Grand Total	2721	4354079.5	2396	3852529.5	

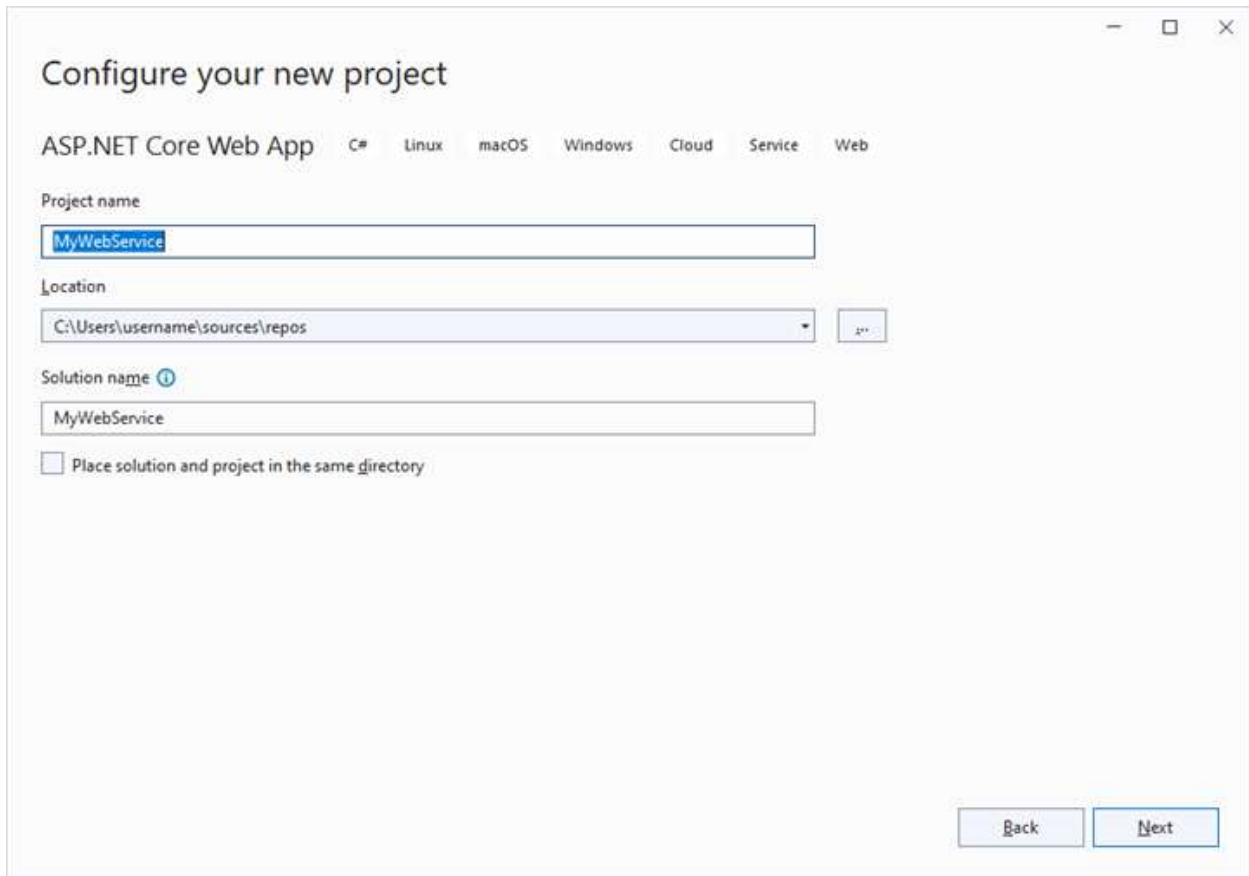
Explore our React Pivot Table sample and ASP.NET Core Web Application to extract data from a MongoDB database and bind to the Pivot Table in [this](#) GitHub repository.

Elasticsearch in EJ2 React Pivotview Component

This section describes how to retrieve data from Elasticsearch database using [Nest](#) library and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch Elasticsearch data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).



Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name

MyWebService

Location

C:\Users\username\source\repos

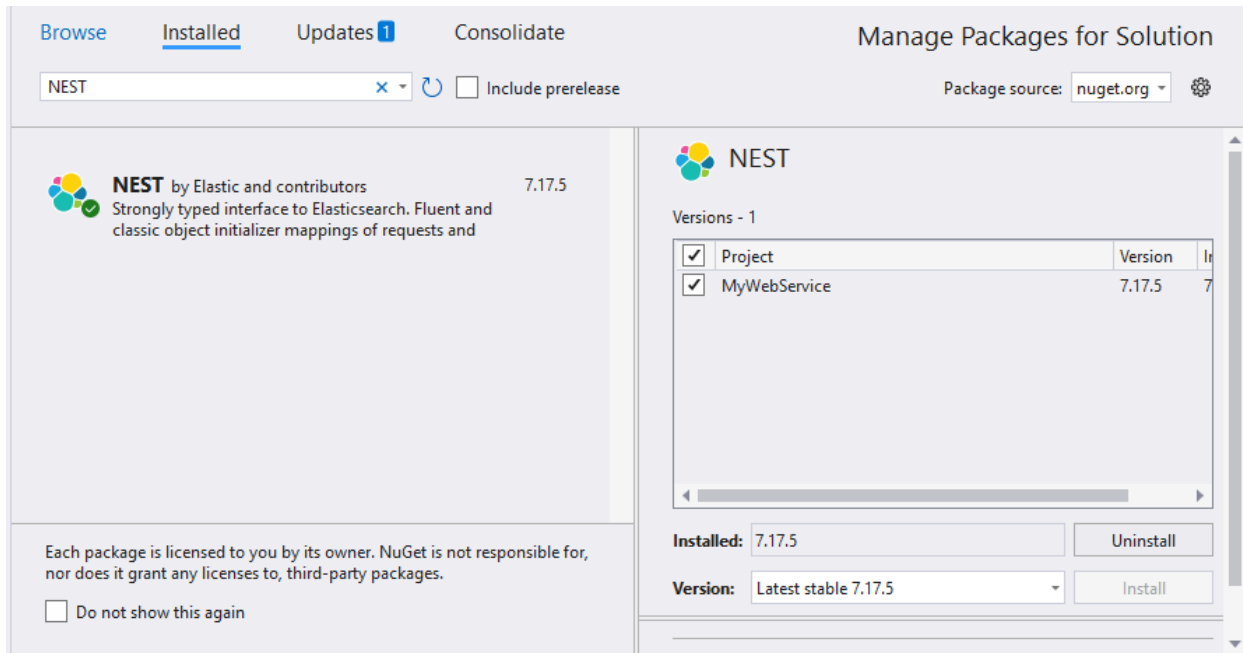
Solution name ⓘ

MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a Elasticsearch Server using the **NEST** in our application, we need to install the [NEST](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **NEST** and install it.



3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **ElasticClient** helps to connect the Elasticsearch database. Next, using **Search** method you can query your Elasticsearch index and retrieve results from the database.

5. In the **Get()** method of the **PivotController.cs** file, the **FetchElasticsearchData** method is used to retrieve the Elasticsearch data, which is then serialized into JSON using **JsonConvert.SerializeObject()**.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Nest;
using Newtonsoft.Json;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetElasticSearchData")]
        public object Get()
        {
            return JsonConvert.SerializeObject(FetchElasticsearchData());
        }
    }
}
```

```

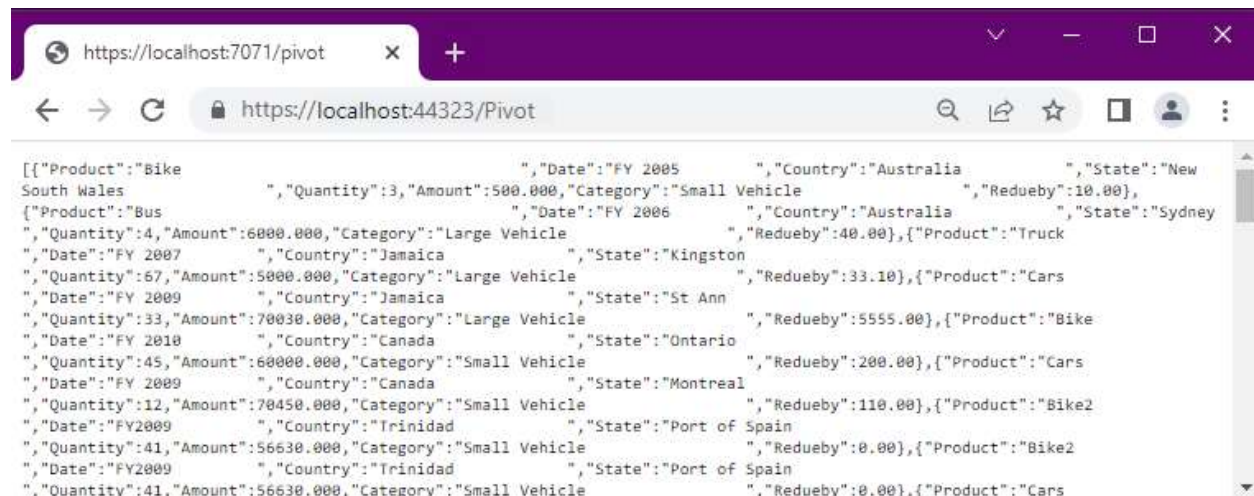
}

private static object FetchElasticsearchData()
{
    // Replace with your own connection string.
    var connectionString = "<Enter your valid connection string here>";
    var uri = new Uri(connectionString);
    var connectionSettings = new ConnectionSettings(uri);
    var client = new ElasticClient(connectionSettings);
    var searchResponse = client.Search<object>{s => s
        .Index("product")
        .Size(1000)
    };
    return searchResponse.Documents;
}
}
}
`

```

6. Run the web application and it will be hosted within the URL <https://localhost:44323>.

7. Finally, the retrieved data from Elasticsearch database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:44323/Pivot>, as shown in the browser page below.



Connecting the Pivot Table to a Elasticsearch database using the Web API service

1. Create a simple React Pivot Table by following the “**Getting Started**” documentation [link](#).

2. Map the hosted Web API's URL link `https://localhost:44323/Pivot` to the Pivot Table component in `app.ts` by using the `url` property under `dataSourceSettings`.

```
`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings = {
    url: 'https://localhost:44323/pivot',
    //Other codes here...
  };
  return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
    showFieldList={true}>
    <Inject services={[FieldList]}></PivotViewComponent>);
};
export default App;
`
```

3. Frame and set the report based on the data retrieved from the Elasticsearch database.

```
`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings = {
    url: 'https://localhost:44323/Pivot',
    expandAll: false,
    enableSorting: true,
    columns: [{ name: 'Product' }],
    values: [{ name: 'Quantity' }, { name: 'Amount', caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'State' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  };
};
```

```

return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}>
<Inject services={[FieldList]}/></PivotViewComponent>);
};
export default App;
`

```

When you run the sample, the resulting pivot table will look like this:

	▶ Large Vehicle		▶ Small Vehicle		Grand Total	
	Quantity	Amount	Quantity	Amount	Quantity	Am
▶ Australia	16	24000	12	2000	28	
▶ Canada			420	1649000	420	
▶ Jamaica	400	300120			400	
▶ Trinidad			328	453040	328	
Grand Total	416	324120	760	2104040	1176	

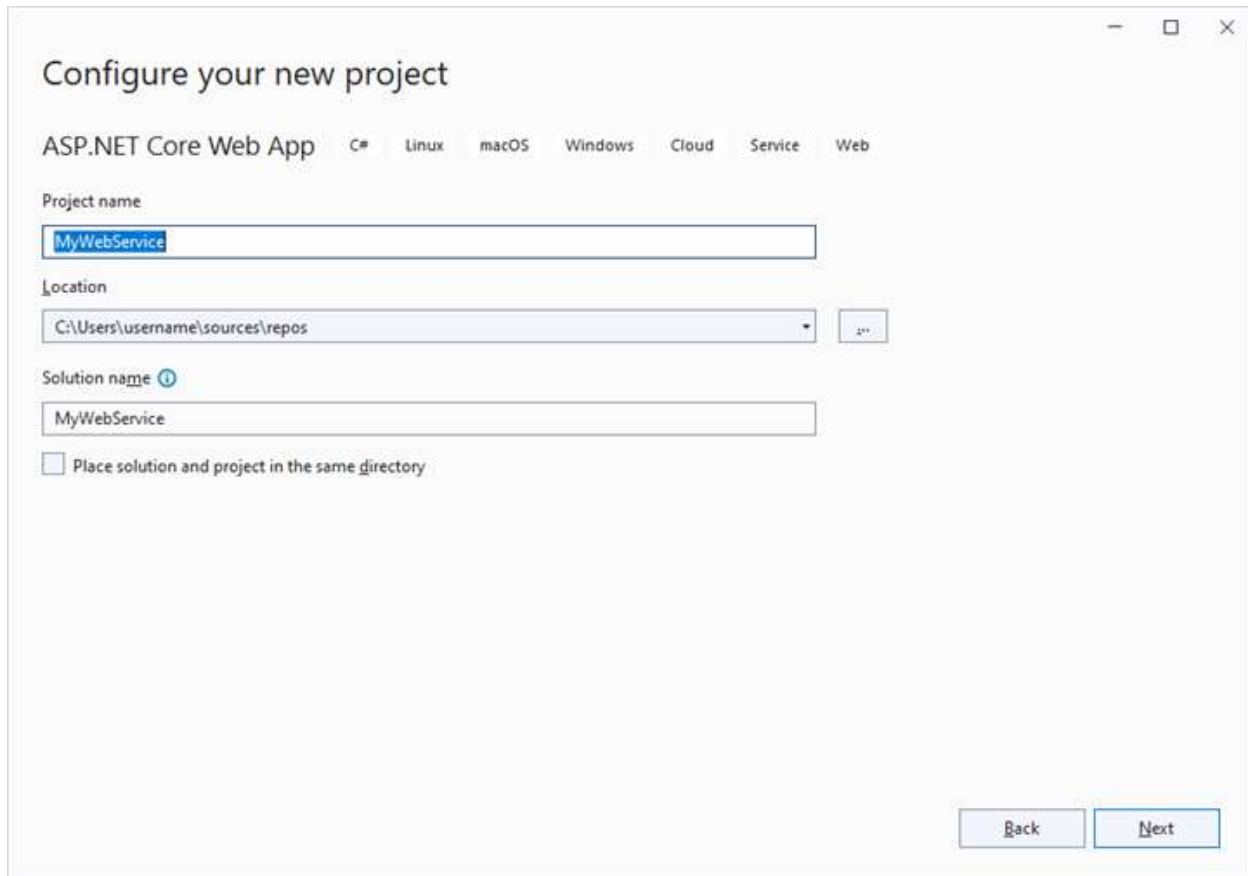
Explore our React Pivot Table sample and ASP.NET Core Web Application to extract data from a Elasticsearch database and bind to the Pivot Table in [this](#) GitHub repository.

Snowflake in EJ2 React Pivotview Component

This section describes how to retrieve data from a Snowflake database using [Snowflake Data](#) and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch Snowflake data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).



Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name
MyWebService

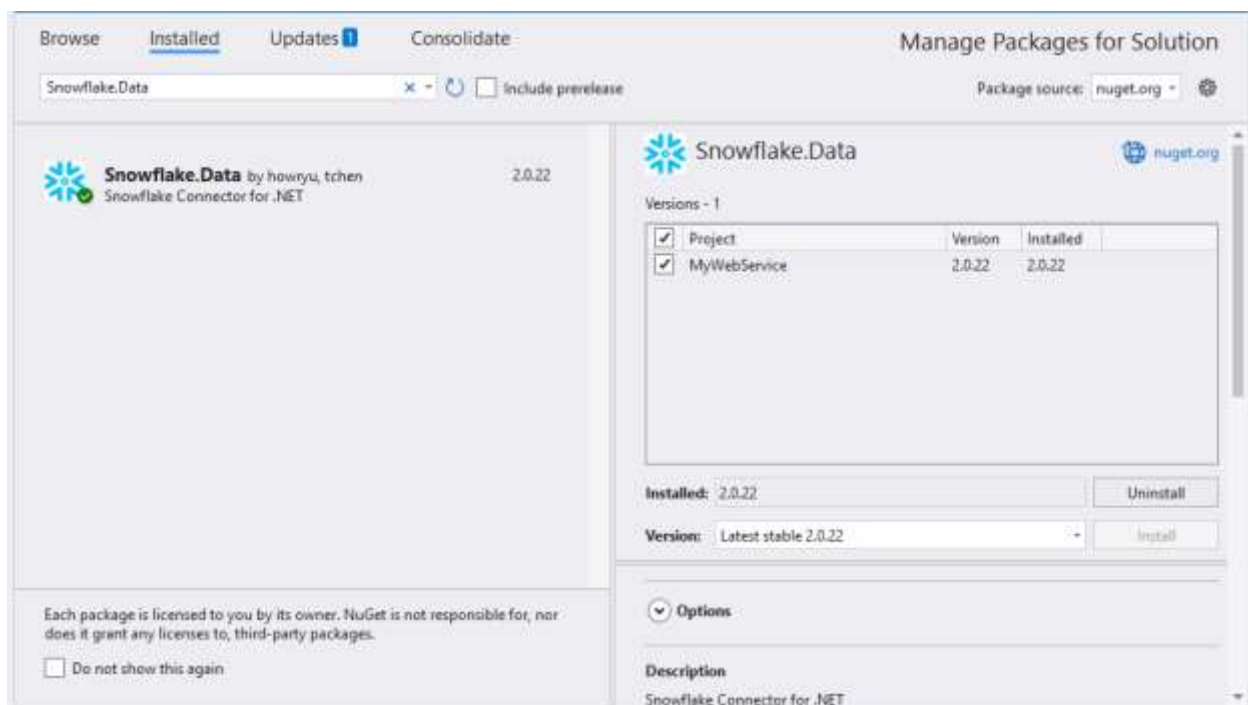
Location
C:\Users\username\source\repos

Solution name
MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a Snowflake Server using the **Snowflake.Data.Client** in our application, we need to install the [Snowflake.Data](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **Snowflake.Data** and install it.



3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **SnowflakeDbConnection** helps to connect the Snowflake database. Next, using **SnowflakeDbDataAdapter** you can process the desired Snowflake query string and retrieve data from the database. The **Fill** method of the **SnowflakeDbDataAdapter** is used to populate the retrieved data into a **DataTable** as shown in the following code snippet.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Snowflake.Data.Client;
using Newtonsoft.Json;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetSnowflakeResult")]
        public object Get()
        {
            return JsonConvert.SerializeObject(FetchSnowflakeResult());
        }
        public static DataTable FetchSnowflakeResult()
        {
            using (SnowflakeDbConnection snowflakeConnection = new SnowflakeDbConnection())
            {
                // Replace with your own connection string.
                snowflakeConnection.ConnectionString = "<Enter your valid connection string here>";
                snowflakeConnection.Open();
                SnowflakeDbDataAdapter adapter = new SnowflakeDbDataAdapter("select * from CALL_CENTER",
                    snowflakeConnection);
                DataTable dataTable = new DataTable();
                adapter.Fill(dataTable);
                snowflakeConnection.Close();
                return dataTable;
            }
        }
    }
}
```

```

}
}
}
}
`

```

5. In the **Get()** method of the **PivotController.cs** file, the **FetchSnowflakeResult** method is used to retrieve the Snowflake data as a **DataTable**, which is then serialized into JSON string using **JsonConvert.SerializeObject()**.

```

`csharp
using Microsoft.AspNetCore.Mvc;
using Snowflake.Data.Client;
using Newtonsoft.Json;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetSnowflakeResult")]
        public object Get()
        {
            return JsonConvert.SerializeObject(FetchSnowflakeResult());
        }
        public static DataTable FetchSnowflakeResult()
        {
            using (SnowflakeDbConnection snowflakeConnection = new SnowflakeDbConnection())
            {
                // Replace with your own connection string.
                snowflakeConnection.ConnectionString = "<Enter your valid connection string here>";
                snowflakeConnection.Open();
                SnowflakeDbDataAdapter adapter = new SnowflakeDbDataAdapter("select * from CALL_CENTER",
                    snowflakeConnection);
                DataTable dataTable = new DataTable();
            }
        }
    }
}

```



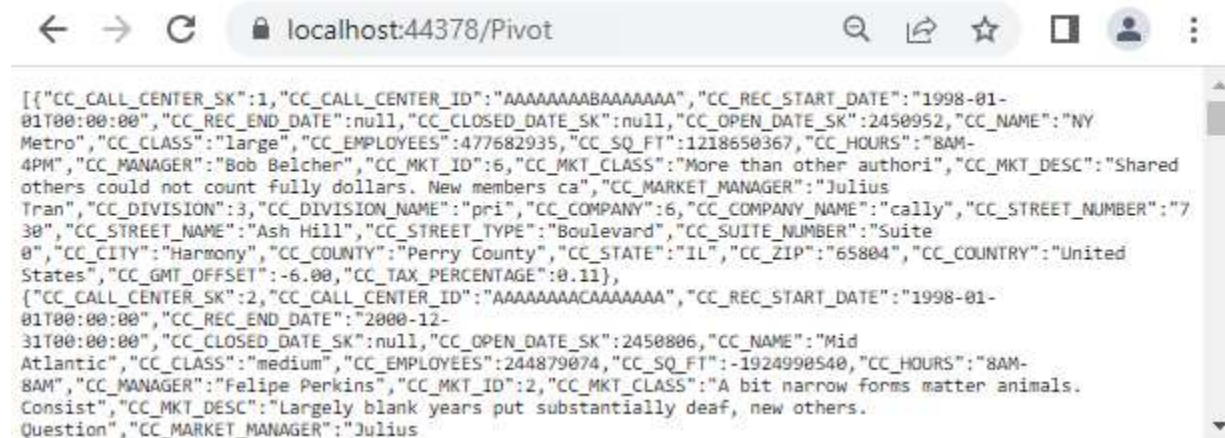
```

adapter.Fill(dataTable);
snowflakeConnection.Close();
return dataTable;
}
}
}
}
,

```

6. Run the application and it will be hosted within the URL <https://localhost:44378/>.

7. Finally, the retrieved data from Snowflake database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:44378/Pivot>, as shown in the browser page below.



Connecting the Pivot Table to a Snowflake database using the Web API service

1. Create a simple React Pivot Table by following the **"Getting Started"** documentation [link](#).
2. Map the hosted Web API's URL link <https://localhost:44378/Pivot> to the Pivot Table component in **app.ts** by using the [url](#) property under [dataSourceSettings](#).

```

`typescript
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
let dataSourceSettings = {
url: 'https://localhost:44378/pivot',
//Other codes here...
};

```

```
return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}>
<Inject services={[FieldList]}/></PivotViewComponent>);
};
export default App;
`
```

3. Frame and set the report based on the data retrieved from the Snowflake database.

`typescript

```
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
let dataSourceSettings = {
url: 'https://localhost:44378/Pivot',
enableSorting: true,
expandAll: false,
dataSource: [],
columns: [
{ name: 'CC_COUNTRY', caption: 'Country' }
],
rows: [
{ name: 'CC_STATE', caption: 'State' },
{ name: 'CC_CITY', caption: 'City' }
],
values: [
{ name: 'CC_COMPANY', caption: 'Company' },
{ name: 'CC_EMPLOYEES', caption: 'Employees' },
{ name: 'CCTAXPERCENTAGE', caption: 'Percentage' },
],
filters: []
};
return (<PivotViewComponent id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}>
<Inject services={[FieldList]}/></PivotViewComponent>);
`
```

```
};
export default App;
`
```

When you run the sample, the resulting pivot table will look like this:

	United States			Grand Total		
	Company	Employees	Percentage	Company	Employees	Percentage
▶ CA	16	413279139	0.4800000000...	16	413279139	0.480000...
▶ CO	5	1160903623	0.18	5	1160903623	
▶ FL	15	1162362540	0.33	15	1162362540	
▶ GA	20	2358562323	0.12	20	2358562323	
▶ IL	8	707852970	0.1699999999...	8	707852970	0.169999...
▶ KS	9	1842349179	0.19	9	1842349179	
▶ LA	6	528468075	0.16	6	528468075	

Explore our React Pivot Table sample and ASP.NET Core Web Application to extract data from a Snowflake database and bind to the Pivot Table in [this](#) GitHub repository.

Olap in React Pivot Table component

Getting started

This section explain steps to create a simple **Pivot Table** with OLAP data source in React environment.

Dependencies

The following list of dependencies are required to use the pivot table component in your application.

```
`javascript
|-- @syncfusion/ej2-react-pivotview
|-- @syncfusion/ej2-pivotview
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-excel-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-pdf-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-charts
|-- @syncfusion/ej2-svg-base
```

```
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-grids
|-- @syncfusion/ej2-react-base
\
```

Setup for Local Development

You can use [create-react-app](#) to setup the application. To install **create-react-app** run the following command.

```
\
npm install -g create-react-app
\
```

To create basic **React** application use following commands.

```
<div class='tsx'>
\
```

```
create-react-app quickstart --scripts-version=react-scripts-ts
\
```

Now, the application is created in the **quickstart** demo folder. Run the following command one-by-one to navigate to the **quickstart** demo folder, and install the required **npm** dependent packages.

```
\
cd quickstart
```

```
npm install
\
```

```
</div>
```

```
<div class='jsx'>
\
```

```
create-react-app quickstart
\
```

Now, the application is cloned in the **quickstart** demo folder. Run the following command one-by-one to navigate to the **quickstart** demo folder, and install the required **npm** dependent packages.

```
cd quickstart
```

```
npm install
```

```
,
```

```
</div>
```

Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) public registry. To install pivot table component, use the following command.

```
,
```

```
npm install @syncfusion/ej2-react-pivotview --save
```

```
,
```

The **--save** will instruct NPM to include the pivot table package inside the **dependencies** section of the **package.json**.

Adding CSS reference

Add pivot table and its [dependent](#) components styles as given below in **src/App.css** file. In this illustration, we have referred **material** theme.

```
`css
```

```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
```

```
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
```

```
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-react-pivotview/styles/material.css';
```

```
,
```

You can also refer other themes like bootstrap, fabric, high-contrast etc. To know about individual component CSS, please refer [here](#).

Next we need to refer **App.css** in the application by importing it in the **src/App.tsx** file as follows.

```
`ts
```

```
import './App.css';
```

```
,
```

Adding pivot table component

You can initialize pivot table component in the application using following steps.

- Import the **PivotViewComponent** (aka, PivotTable) component from the **@syncfusion/ej2-react-pivotview** package in **app.ts** file.
- Then you can initialize pivot table component () using following code.

```
`ts
import { IDataOptions, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  return <PivotViewComponent/>
};
export default App;
`
```

After initialization, add the the following code in **src/App.tsx** file to populate pivot table with a sample OLAP data source. Refer [here](#) to know the more details about OLAP data binding.

```
`ts
import { IDataOptions, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings: IDataOptions = {
    catalog: 'Adventure Works DW 2008 SE',
    cube: 'Adventure Works',
    enableSorting: true,
    localeIdentifier: 1033,
    providerType: 'SSAS',
    url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350}
    dataSourceSettings={dataSourceSettings} />)
};
export default App;
```

```

ReactDOM.render(<App />, document.getElementById('sample'));
,
`ts
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import './App.css';
function App() {
  let dataSourceSettings = {
    catalog: 'Adventure Works DW 2008 SE',
    cube: 'Adventure Works',
    enableSorting: true,
    localeIdentifier: 1033,
    providerType: 'SSAS',
    url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350}
    dataSourceSettings={dataSourceSettings}/>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
,

```

Adding OLAP cube elements to row, column, value and filter axes

Now that pivot table is initialized and assigned with sample OLAP data source, will further move to showcase the component by organizing appropriate [OLAP cube elements](#) in [rows](#), [columns](#), [values](#) and [filters](#) axes.

In [dataSourceSettings](#) property, four major axes [rows](#), [columns](#), [values](#) and [filters](#) plays a vital role in defining and organizing [OLAP cube elements](#) from the bound data source, to render the entire pivot table component in a desired format.

[rows](#) – Collection of [OLAP cube elements](#) (such as Hierarchies, NamedSet, Calculated Members etc.,) that needs to be displayed in row axis of the pivot table.

[columns](#) – Collection of [OLAP cube elements](#) (such as Hierarchies, NamedSet, Calculated Members etc.,) that needs to be displayed in column axis of the pivot table.

[values](#) – Collection of [OLAP cube elements](#) (such as Measures, Calculated Measures) that needs to be displayed as aggregated numeric values in the pivot table.

[filters](#) - Collection of [OLAP cube elements](#) (such as Hierarchies and Calculated Members) that would act as master filter over the data bound in row, column and value axes of the pivot table.

In-order to define each [OLAP cube element](#) in the respective axis, the following basic properties should be set.

- [name](#): It allows to set the unique name of the hierarchies, named set, measures, calculated members etc., from the bound OLAP data source. It's casing should match exactly like in the data source and if not set properly, the pivot table will be rendered as empty.
- [caption](#): It allows to set the caption, which is the alias name of the unique name that needs to be displayed in the pivot table. If not provided, unique name will be displayed.

In this sample, "Product Categories" is added in column, "Customer Geography" in row, and "Customer Count" and "Internet Sales Amount" in value axes respectively.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
  let dataSourceSettings = {
    catalog: 'Adventure Works DW 2008 SE',
    cube: 'Adventure Works',
    providerType: 'SSAS',
    enableSorting: true,
    url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
    localeIdentifier: 1033,
    rows: [
      { name: '[Customer].[Customer Geography]', caption: 'Customer
Geography' },
    ],
    columns: [
      { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
      { name: '[Measures]', caption: 'Measures' },
    ],
    values: [
      { name: '[Measures].[Customer Count]', caption: 'Customer Count' },
      { name: '[Measures].[Internet Sales Amount]', caption: 'Internet
Sales Amount' }
    ]
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}>
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX


```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ]
    };
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}>
        </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Applying formatting to measures

Formatting defines a way in which values should be displayed in pivot table. For example, format **"C0"** denotes the values should be displayed in currency pattern without decimal points. To do so, define the [formatSettings](#) with its [name](#) and [format](#) properties. In this sample, the [name](#) property is set as "[Measures].[Internet Sales Amount]", a measure from value axis and its format is set as "C0". Likewise, we can set format for other measures as well.

Only measures from value axis, which is in the form of numeric data values are applicable for formatting.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',

```

```

        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption: 'Customer
Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer Count'
},
            { name: '[Measures].[Internet Sales Amount]', caption: 'Internet
Sales Amount' }
        ],
        formatSettings: [{ name: '[Measures].[Internet Sales Amount]',
format: 'C0' }]
    };
    let pivotObj;
    return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}>
    </PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [

```

```

        { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
        { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
    ],
    formatSettings: [{ name: '[Measures].[Internet Sales Amount]',
format: 'C0' }]
    };
    let pivotObj: PivotViewComponent;
    return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}>
    </PivotViewComponent>
    };
    export default App;
    ReactDOM.render(<App />, document.getElementById('sample'));

```

Enable grouping bar

The Grouping Bar feature automatically populates [OLAP cube elements](#) from the bound data source and allows end users to drag [OLAP cube elements](#) between different axes such as [rows](#), [columns](#), [values](#) and [filters](#), and change pivot view at runtime. Sorting, filtering and removing of elements is also possible. It can be enabled by setting the [showGroupingBar](#) property to **true** and by injecting the **GroupingBar** module as follows.

If the **GroupingBar** module is not injected, the grouping bar will not be rendered with the pivot table component.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Inject, PivotViewComponent, GroupingBar } from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption: 'Customer
Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer Count' },
            { name: '[Measures].[Internet Sales Amount]', caption: 'Internet
Sales Amount' }
        ],
    };

```

```

        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items: ['[Date].[Fiscal].[Fiscal
Quarter].&[2002]&[4]',
                '[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    let pivotObj;
    return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showGroupingBar={true}><Inject services={[GroupingBar]}/>
    </PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { IDataOptions, IDataset, Inject, PivotViewComponent, GroupingBar }
from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {

```

```

        name: '[Date].[Fiscal]', items:
        ['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
        '[Date].[Fiscal].[Fiscal Year].&[2005]'],
        levelCount: 3
    }
    ]
};
let pivotObj: PivotViewComponent;
return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showGroupingBar={true}><Inject services={[GroupingBar]} />
</PivotViewComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Enable pivot field list

The component provides a built-in Field List similar to Microsoft Excel. It allows you to add or remove [OLAP cube elements](#) and also rearrange the [OLAP cube elements](#) between different axes, including [rows](#), [columns](#), [values](#) and [filters](#) along with filter and sort options dynamically at runtime. It can be enabled by setting the [showFieldList](#) property to **true** and by injecting the **FieldList** module as follows.

If the **FieldList** module is not injected, the Field List will not be rendered with the pivot table component.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption: 'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer Count' },
            { name: '[Measures].[Internet Sales Amount]', caption: 'Internet Sales Amount' },
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
    };
}

```

```

        filterSettings: [
            {
                name: '[Date].[Fiscal]', items: ['[Date].[Fiscal].[Fiscal
Quarter].&[2002]&[4]',
                '[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}/>
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
                '[Date].[Fiscal].[Fiscal Year].&[2005]'],
            }
        ]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}/>
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

```

        levelCount: 3
      }
    ]
  };
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Exploring filter axis

The filter axis contains collection of [OLAP cube elements](#) such as hierarchies and calculated members that would act as master filter over the data bound in [rows](#), [columns](#) and [values](#) axes of the pivot table. The [OLAP cube elements](#) along with filter members could be set to filter axis either through report via code behind or by dragging and dropping [OLAP cube elements](#) from other axes to filter axis via grouping bar or field list at runtime.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
  let dataSourceSettings = {
    catalog: 'Adventure Works DW 2008 SE',
    cube: 'Adventure Works',
    providerType: 'SSAS',
    enableSorting: true,
    url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
    localeIdentifier: 1033,
    rows: [
      { name: '[Customer].[Customer Geography]', caption: 'Customer Geography' },
    ],
    columns: [
      { name: '[Product].[Product Categories]', caption: 'Product Categories' },
      { name: '[Measures]', caption: 'Measures' },
    ],
    values: [
      { name: '[Measures].[Customer Count]', caption: 'Customer Count' },
      { name: '[Measures].[Internet Sales Amount]', caption: 'Internet Sales Amount' },
    ],
    filters: [
      { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
    ],
    filterSettings: [
      {

```

```

        name: '[Date].[Fiscal]', items: ['[Date].[Fiscal].[Fiscal
Quarter].&[2002]&[4]',
        '[Date].[Fiscal].[Fiscal Year].&[2005]'],
        levelCount: 3
    }
]
};
let pivotObj;
return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}>/>
</PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
                '[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
}

```



```

    ]
  };
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Calculated field

The calculated field allows user to insert or add a new calculated field based on the available [OLAP cube elements](#) from the bound data source. Calculated fields are nothing but customized dimensions or measures that are newly created based on the user-defined expression.

The two types of calculated fields are as follows:

- **Calculated Measure** – Creates a new measure through user-defined expression.
- **Calculated Dimension** – Creates a new dimension through user-defined expression.

It can be customized using the [calculatedFieldsSettings](#) property through code behind. The setting required for calculate field feature at code behind are:

- [name](#): It allows to set the unique name for new calculated field.
- [formula](#): It allows to set the user-defined expression.
- [hierarchyUniqueName](#): It allows to specify dimension unique name whose hierarchies alone should be used in the expression. This will be applicable only for calculated dimension.
- [formatString](#): It allows to set the format string for the resultant calculated field.

You need to set [isCalculatedField](#) property to true, while adding calculated fields to respective axis through code behind.

Also calculated fields can be added at run time through the built-in dialog. The dialog can be enabled by setting the [allowCalculatedField](#) property to **true** and by injecting the **CalculatedField** module as follows. You will see a button enabled in the Field List UI automatically to invoke the calculated field dialog and perform necessary operation.

If the **CalculatedField** module is not injected, the calculated field dialog will not be rendered with the pivot table component. Moreover calculated measure can be added only in value axis.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { CalculatedField, FieldList, Inject, PivotViewComponent } from
'@syncfusion/ej2-react-pivotview';
function App() {
  let dataSourceSettings = {
    catalog: 'Adventure Works DW 2008 SE',
    cube: 'Adventure Works',
    providerType: 'SSAS',
    enableSorting: true,

```

```

    url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
    localeIdentifier: 1033,
    rows: [
      { name: '[Customer].[Customer Geography]', caption: 'Customer
Geography' },
    ],
    columns: [
      { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
      { name: '[Measures]', caption: 'Measures' },
    ],
    values: [
      { name: '[Measures].[Customer Count]', caption: 'Customer Count'
},
      { name: '[Measures].[Internet Sales Amount]', caption: 'Internet
Sales Amount' },
      { name: 'Order on Discount', isCalculatedField: true }
    ],
    filters: [
      { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
    ],
    calculatedFieldSettings: [
      {
        name: 'BikeAndComponents',
        formula: '([Product].[Product Categories].[Category].[Bikes]
+ [Product].[Product Categories].[Category].[Components] )',
        hierarchyUniqueName: '[Product].[Product Categories]',
        formatString: 'Standard'
      },
      {
        name: 'Order on Discount',
        formula: '[Measures].[Order Quantity] + ([Measures].[Order
Quantity] * 0.10)',
        formatString: 'Currency'
      }
    ],
    filterSettings: [
      {
        name: '[Date].[Fiscal]', items: ['[Date].[Fiscal].[Fiscal
Quarter].&[2002]&[4]',
        '[Date].[Fiscal].[Fiscal Year].&[2005]'],
        levelCount: 3
      }
    ]
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}><Inject
services={[CalculatedField, FieldList]}>
  </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { CalculatedField, FieldList, IDataOptions, IDataset, Inject,
PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' },
            { name: 'Order on Discount', isCalculatedField: true }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        calculatedFieldSettings: [
            {
                name: 'BikeAndComponents',
                formula: '([Product].[Product
Categories].[Category].[Bikes] + [Product].[Product
Categories].[Category].[Components] )',
                hierarchyUniqueName: '[Product].[Product Categories]',
                formatString: 'Standard'
            },
            {
                name: 'Order on Discount',
                formula: '[Measures].[Order Quantity] +
([Measures].[Order Quantity] * 0.10)',
                formatString: 'Currency'
            }
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };

```

```

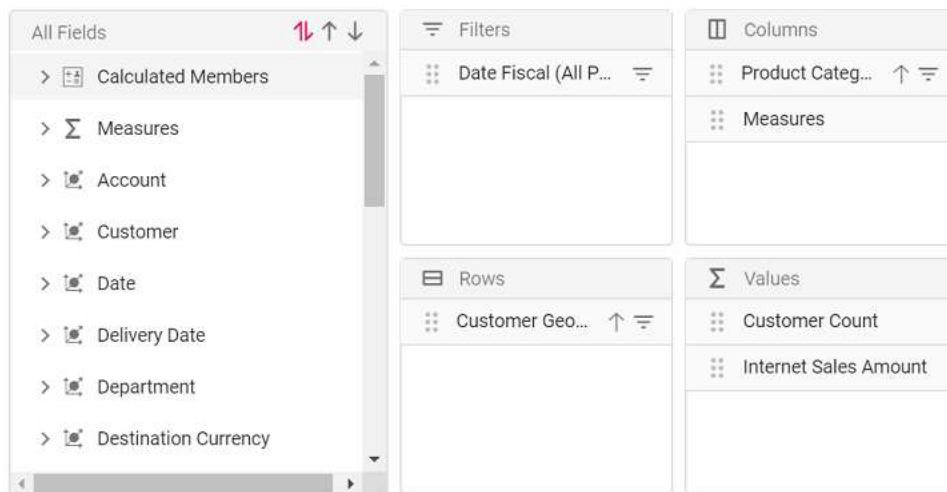
    ];
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}><Inject
services={[CalculatedField, FieldList]}/>
    </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Users can add a calculated field at runtime through the built-in dialog by using the following steps.

Step 1: Click the "CALCULATED FIELD" button in the field list dialog positioned at the top right corner. The calculated field dialog will be opened now. Enter the name of the calculated field to be created.

Field List



CLOSE

Create Calculated Field

All Fields

- > Calculated Members
- > Measures
- > Account
- > Customer
- > Date
- > Delivery Date
- > Department
- > Destination Currency
- > Employee
- > Geography
- > Internet Sales Order Details
- > Organization
- > Product

Field Name
BikeAndComponents

Expression
Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type
Measure

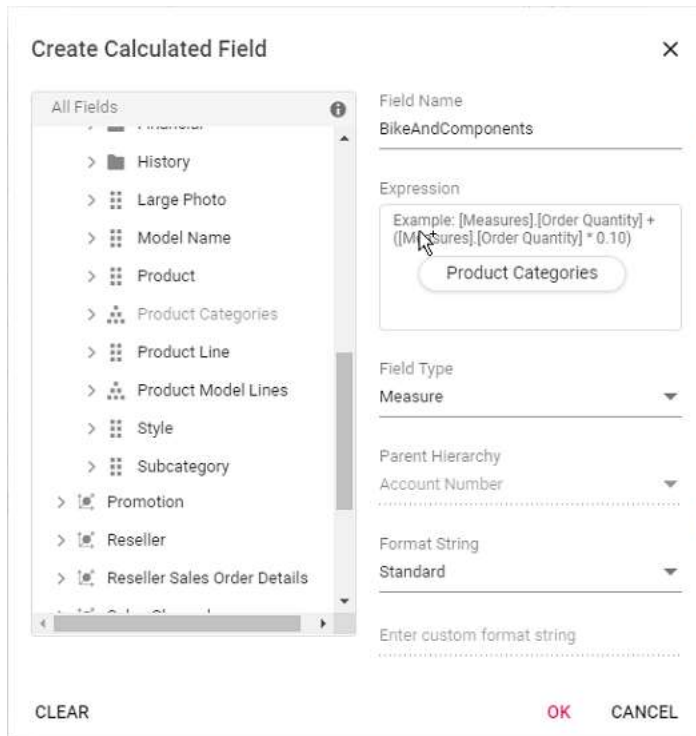
Parent Hierarchy
Account Number

Format String
Standard

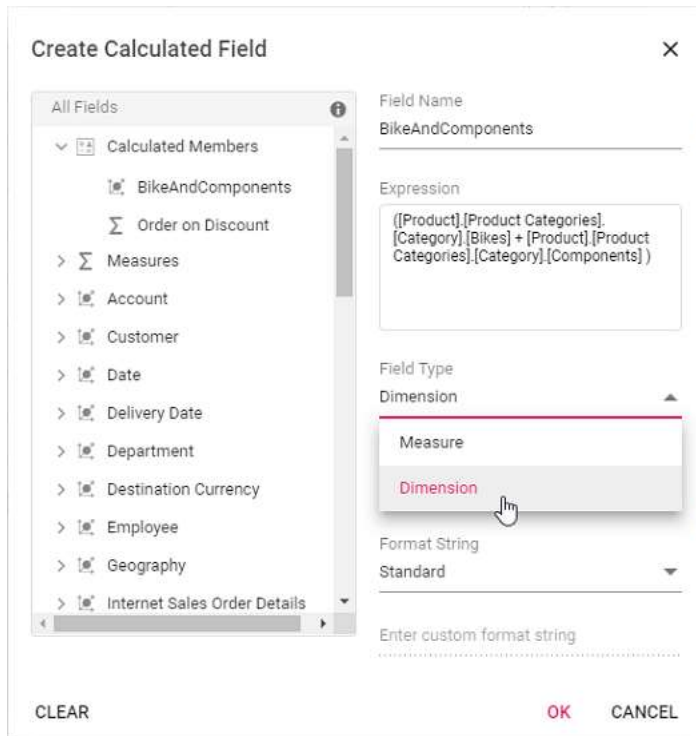
Enter custom format string

CLEAR OK CANCEL

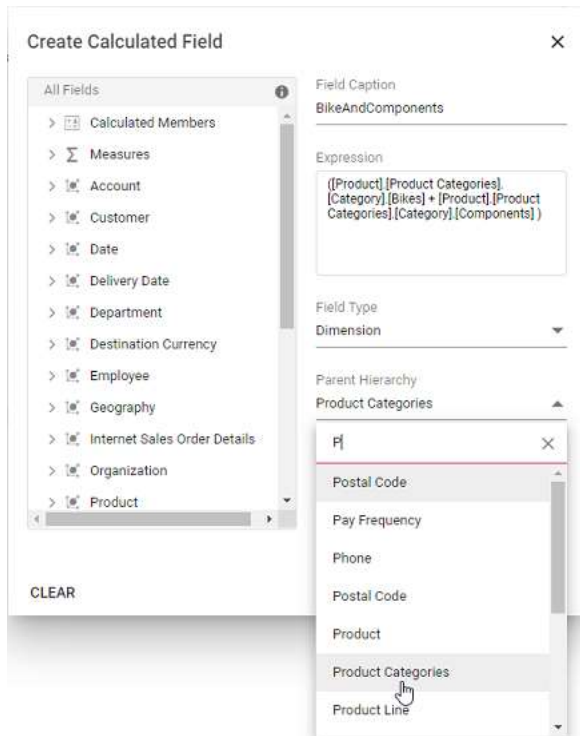
Step 2: Frame the expression by dragging and dropping the fields from the tree view on the left side of the dialog using simple arithmetic operators. **Example:** "IIF([Measures].[Internet Sales Amount]^0.5 > 100, [Measures].[Internet Sales Amount]*100, [Measures].[Internet Sales Amount]/100)". Please refer here to learn more about the supported [operators](#) and [functions](#) to frame the expression.



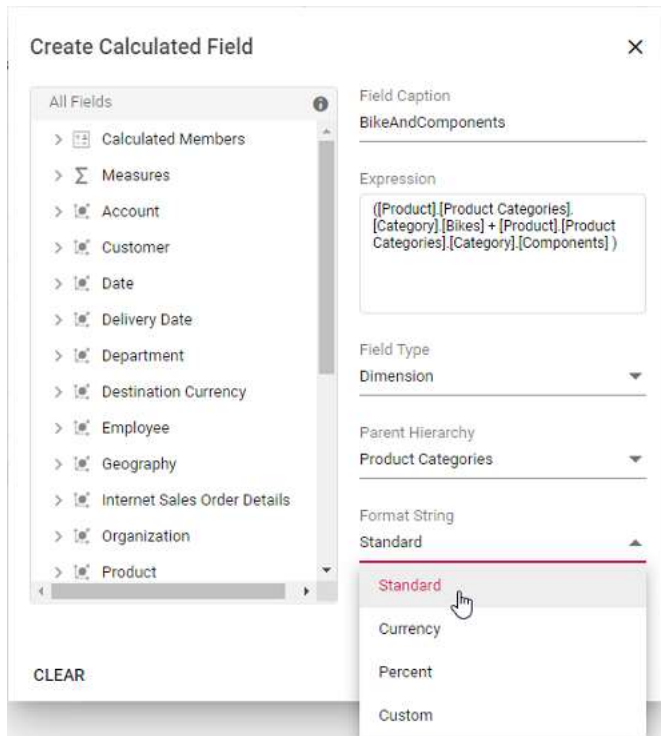
Step 3: Confirm the type of the field to be created - calculated measure or calculated dimension.



Step 4: Choose the parent hierarchy of the calculated field. NOTE: It is only applicable to the calculated dimension.



Step 5: Then select the format string from the drop-down list and finally click "OK".



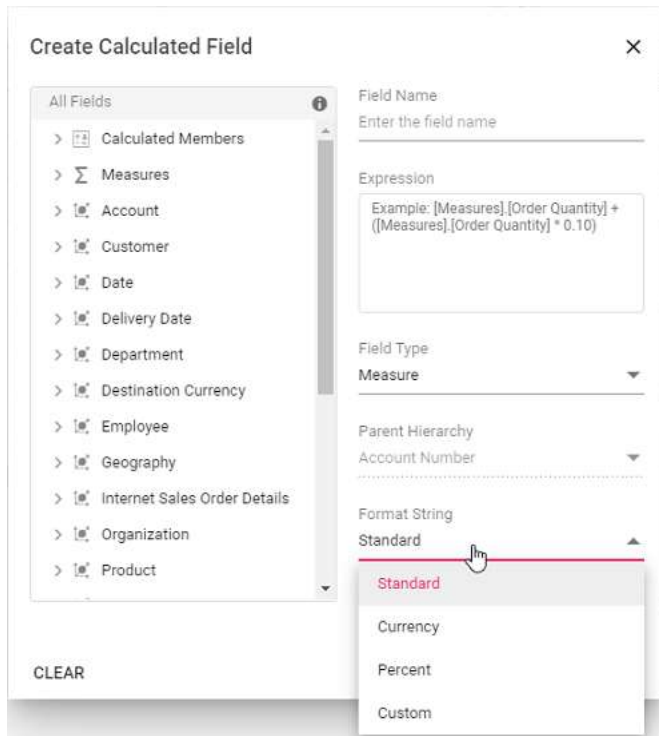
	Accessories		
	Customer Count	Internet Sales Amount	Order on Discount
► Australia	2,905	\$138,690.63	\$68,124.10
► Canada	1,230	\$103,377.85	\$68,124.10
► France	1,505	\$63,406.78	\$68,124.10
► Germany	1,535	\$62,232.59	\$68,124.10

Format String

Allows you to specify the required format string while creating new calculated field. Supported format strings are:

- **Standard** - Denotes the numeric type.
- **Currency** - Denotes the currency type.
- **Percent** - Denotes the percentage type.
- **Custom** - Denotes the custom format. For example: "###0.##0#". This shows the value "9584.3" as "9584.300."

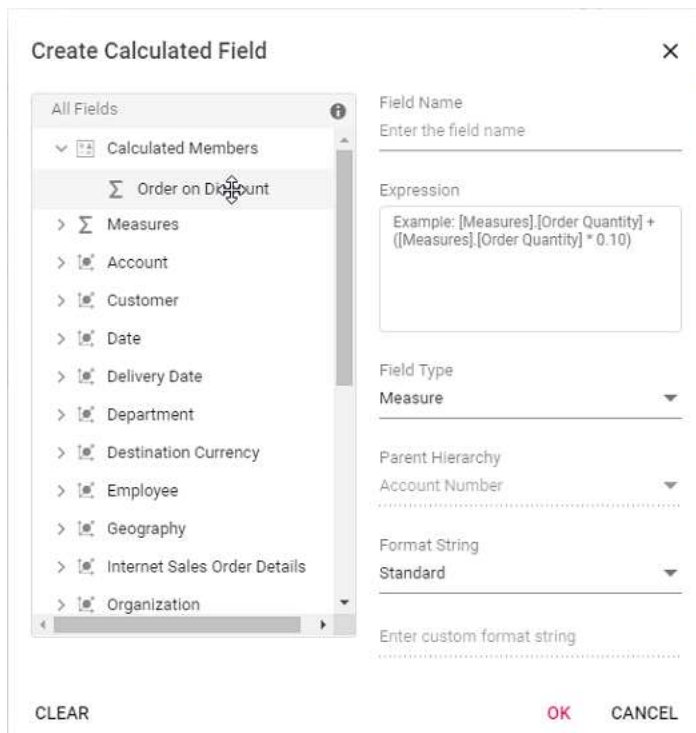
By default, **Standard** will be selected from the drop down list.



Renaming the existing calculated field

Existing calculated field can be renamed only through the UI at runtime. To do so, open the calculated field dialog, click the target field. User can now see the existing name getting displayed in the text box at the top of the dialog. Now, change the name based on user requirement and click "OK".

<!-- markdownlint-disable MD012 -->



Create Calculated Field

All Fields:

- Calculated Members
 - Order on Discount
- Measures
- Account
- Customer
- Date
- Delivery Date
- Department
- Destination Currency
- Employee
- Geography
- Internet Sales Order Details
- Organization

Field Caption: Order Quantity on Discount

Expression: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type: Measure

Parent Hierarchy: Account Number

Format String: Standard

Enter custom format string

CLEAR **OK** **CANCEL**

[Editing the existing calculated field formula](#)

Existing calculated field formula can be edited only through the UI at runtime. To do so, open the calculated field dialog, click the target field. User can now see the existing expression getting displayed in a "Expression" section. Now, change the expression based on user requirement and click "OK".

Create Calculated Field

All Fields

Calculated Members

Σ Order on Discount

Σ Measures

Account

Customer

Date

Delivery Date

Department

Destination Currency

Employee

Geography

Internet Sales Order Details

Organization

Field Name

Enter the field name

Expression

Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type

Measure

Parent Hierarchy

Account Number

Format String

Standard

Enter custom format string

CLEAR

OK

CANCEL

Create Calculated Field

All Fields

Calculated Members

Σ Order on Discount

Σ Measures

Account

Customer

Date

Delivery Date

Department

Destination Currency

Employee

Geography

Internet Sales Order Details

Organization

Field Caption

Order Quantity on Discount

Expression

[Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.50)

Field Type

Measure

Parent Hierarchy

Account Number

Format String

Standard

Enter custom format string

CLEAR

OK

CANCEL

Reusing the existing formula in a new calculate field

While creating a new calculated field, if user wants to add the formula of an existing calculated field, it can be done easily. To do so, simply drag-and-drop the existing calculated field to the "Expression" section.

Create Calculated Field

All Fields

- Calculated Members
 - Order on Discount
- Measures
- Account
- Customer
- Date
- Delivery Date
- Department
- Destination Currency
- Employee
- Geography
- Internet Sales Order Details
- Organization

Field Name
Enter the field name

Expression
Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type
Measure

Parent Hierarchy
Account Number

Format String
Standard

Enter custom format string

CLEAR OK CANCEL

Create Calculated Field

All Fields

- Calculated Members
 - Order on Discount**
- Measures
- Account
- Customer
- Date
- Delivery Date
- Department
- Destination Currency
- Employee
- Geography
- Internet Sales Order Details
- Organization

Field Caption
Discount Quantity

Expression
Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)
Order on Discount

Field Type
Measure

Parent Hierarchy
Account Number

Format String
Standard

Enter custom format string

CLEAR OK CANCEL

Create Calculated Field

All Fields

- Calculated Members
 - Order on Discount**
- Measures
- Account
- Customer
- Date
- Delivery Date
- Department
- Destination Currency
- Employee
- Geography
- Internet Sales Order Details
- Organization

Field Caption
Discount Quantity

Expression
[Measures].[Order on Discount]

Field Type
Measure

Parent Hierarchy
Account Number

Format String
Standard

Enter custom format string

CLEAR OK CANCEL

Modifying the existing format string

Existing calculated field's format string can be modified only through the UI at runtime. To do so, open the calculated field dialog and click the target calculated field. User can now see the format string for

the existing calculated field getting displayed in a drop-down list. Change the format string based on the requirement and finally click "OK".

Create Calculated Field

All Fields

- Calculated Members
 - Order on Discount**
- Measures
- Account
- Customer
- Date
- Delivery Date
- Department
- Destination Currency
- Employee
- Geography
- Internet Sales Order Details
- Organization

Field Name
Enter the field name

Expression
Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type
Measure

Parent Hierarchy
Account Number

Format String
Standard

Enter custom format string

CLEAR OK CANCEL

Create Calculated Field

All Fields

- Calculated Members
 - Order on Discount**
- Measures
- Account
- Customer
- Date
- Delivery Date
- Department
- Destination Currency
- Employee
- Geography
- Internet Sales Order Details
- Organization

Field Caption
Order on Discount

Expression
[Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type
Measure

Parent Hierarchy
Account Number

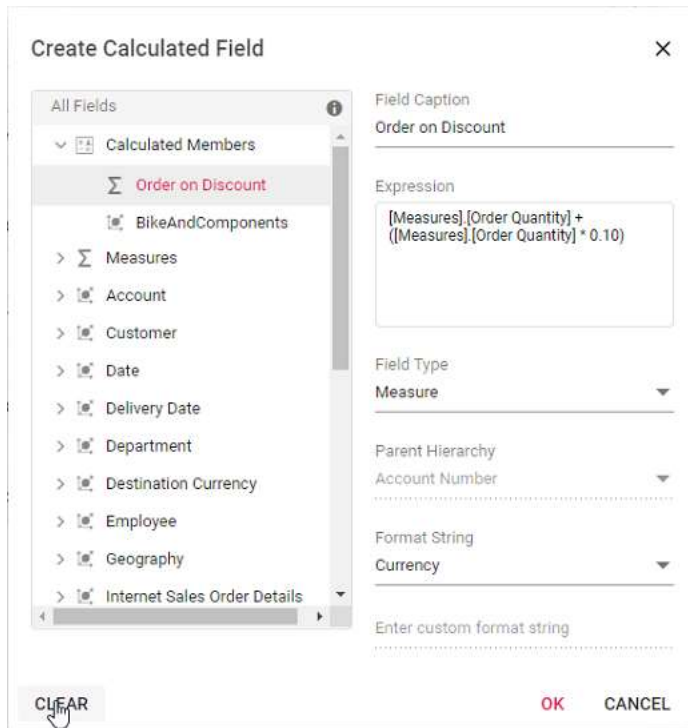
Format String
Currency

Standard
Currency
Percent
Custom

CLEAR

Clearing the changes while editing the calculated field

Previous changes can be cleared by using the "Clear" option while performing operations such as creating and editing the calculated field. To do so, click the "Clear" button in the bottom left corner of the dialog.



Virtual Scrolling

Allows large amounts of data to be loaded without any performance degradation by rendering rows and columns in relation to the current viewport. Rest of the data will be brought into the viewport dynamically based on vertical or horizontal scroll position. This feature can be enabled by setting the [enableVirtualization](#) property to **true**.

To use the virtual scrolling feature, inject the **VirtualScroll** module into the pivot table.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent, Inject, VirtualScroll } from '@syncfusion/ej2-react-pivotview';
function App() {
  let dataSourceSettings = {
    catalog: 'Adventure Works DW 2008 SE',
    cube: 'Adventure Works',
    providerType: 'SSAS',
    enableSorting: true,
    url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
    localeIdentifier: 1033,
    rows: [
      { name: '[Customer].[Customer]', caption: 'Customer' },
    ],
    columns: [
```



```

        { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
        { name: '[Measures]', caption: 'Measures' },
    ],
    values: [
        { name: '[Measures].[Customer Count]', caption: 'Customer Count'
    },
        { name: '[Measures].[Internet Sales Amount]', caption: 'Internet
Sales Amount' }
    ],
    formatSettings: [{ name: '[Measures].[Internet Sales Amount]',
format: 'C0' }]
    };
    let pivotObj;
    return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enableVirtualization={true}
dataSourceSettings={dataSourceSettings}><Inject
services={[VirtualScroll]}/></PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { IDataOptions, IDataset, PivotViewComponent, Inject, VirtualScroll }
from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer]', caption: 'Customer' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        formatSettings: [{ name: '[Measures].[Internet Sales Amount]',
format: 'C0' }]
    };
    let pivotObj: PivotViewComponent;

```

```

    return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
    height={350} enableVirtualization={true}
    dataSourceSettings={dataSourceSettings}><Inject
    services={[VirtualScroll]}></PivotViewComponent>
    </>;
    export default App;
    ReactDOM.render(<App />, document.getElementById('sample'));

```

Limitations for virtual scrolling

- In virtual scrolling, the [columnWidth](#) property in [gridSettings](#) should be in pixels, and percentage values are not accepted.
- Resizing columns or setting width to individual columns affects the calculation used to pick the correct page on scrolling.
- When using OLAP data, subtotals and grand totals are only displayed when measures are bound at the last position in the [rows](#) or [columns](#) axis. Otherwise, the data from the pivot table will be shown without summary totals.
- When the pivot table's width and height are large, the loading data count in the current, previous, and next viewports (pages) will also increase, affecting performance.

Run the application

The quickstart project is configured to compile and run the application in the browser. Use the following command to run the application.

`

npm start

`

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { CalculatedField, FieldList, Inject, PivotViewComponent } from
'@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption: 'Customer
Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [

```

```

        { name: '[Measures].[Customer Count]', caption: 'Customer Count'
    },
        { name: '[Measures].[Internet Sales Amount]', caption: 'Internet
Sales Amount' },
        { name: 'Order on Discount', isCalculatedField: true }
    ],
    filters: [
        { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
    ],
    calculatedFieldSettings: [
        {
            name: 'BikeAndComponents',
            formula: '([Product].[Product Categories].[Category].[Bikes]
+ [Product].[Product Categories].[Category].[Components] )',
            hierarchyUniqueName: '[Product].[Product Categories]',
            formatString: 'Standard'
        },
        {
            name: 'Order on Discount',
            formula: '[Measures].[Order Quantity] + ([Measures].[Order
Quantity] * 0.10)',
            formatString: 'Currency'
        }
    ],
    filterSettings: [
        {
            name: '[Date].[Fiscal]', items: ['[Date].[Fiscal].[Fiscal
Quarter].&[2002]&[4]',
            '[Date].[Fiscal].[Fiscal Year].&[2005]'],
            levelCount: 3
        }
    ]
};
let pivotObj;
return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}><Inject
services={[CalculatedField, FieldList]}/>
</PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { CalculatedField, FieldList, IDataOptions, IDataset, Inject,
PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
    };
}

```

```

        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' },
            { name: 'Order on Discount', isCalculatedField: true }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        calculatedFieldSettings: [
            {
                name: 'BikeAndComponents',
                formula: '([Product].[Product
Categories].[Category].[Bikes] + [Product].[Product
Categories].[Category].[Components] )',
                hierarchyUniqueName: '[Product].[Product Categories]',
                formatString: 'Standard'
            },
            {
                name: 'Order on Discount',
                formula: '[Measures].[Order Quantity] +
([Measures].[Order Quantity] * 0.10)',
                formatString: 'Currency'
            }
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}><Inject
services={ [CalculatedField, FieldList] }/>
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Data Binding

To bind OLAP datasource to the pivot table, you need to specify following properties under [dataSourceSettings](#) option.

Properties	Description
cube	Points the respective cube name from OLAP database.
providerType	Points the provider type for pivot table to identify the type of data source.
url	Contains the cube URL for establishing the connection (online).
catalog	Contains the database name (catalog name) to fetch the data.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption: 'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer Count' },
            { name: '[Measures].[Internet Sales Amount]', caption: 'Internet Sales Amount' },
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items: ['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]', '[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    let pivotObj;
```

```

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}/>
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]} />
    </PivotViewComponent>);
};

```

```
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Fields

Measures in row axis

By default, the measures are plotted in column axis. To plot those measures in row axis, place the **Measures** button in the row axis as follows.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption: 'Customer Geography' },
            { name: '[Measures]', caption: 'Measures' }
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product Categories' }
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer Count' },
            { name: '[Measures].[Internet Sales Amount]', caption: 'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' }
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items: ['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
                    '[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
        height={350} dataSourceSettings={dataSourceSettings}
        showFieldList={true}><Inject services={[FieldList]}/>
        </PivotViewComponent>);
}
```

```
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
            { name: '[Measures]', caption: 'Measures' }
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' }
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```


Measures in different position

You can place measures in different position in row or column axis either thorough code behind or UI. In this sample, **measures** placed before the dimension in the column axis.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption: 'Customer Geography' },
        ],
        columns: [
            { name: '[Measures]', caption: 'Measures' },
            { name: '[Product].[Product Categories]', caption: 'Product Categories' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer Count' },
            { name: '[Measures].[Internet Sales Amount]', caption: 'Internet Sales Amount' },
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items: ['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]', '[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}><Inject services={[FieldList]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Measures]', caption: 'Measures' },
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Named set

Named set is a multidimensional expression (MDX) that returns a set of dimension members, which can be created by combining the cube data, arithmetic operators, numbers, and functions.

You can bind the named sets in the pivot table by setting its unique name in the [name](#) property either in row or column axis and [isNamedSet](#) boolean property to **true**. In this sample, we have added "Core Product Group" named set in the column axis.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption: 'Customer Geography' },
        ],
        columns: [
            {
                name: "[Core Product Group]",
                isNamedSet: true
            },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer Count' },
            { name: '[Measures].[Internet Sales Amount]', caption: 'Internet Sales Amount' },
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items: ['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]', '[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}><Inject services={[FieldList]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';

```

```

import * as ReactDOM from 'react-dom';
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
function App() {
    let dataSourceSettings: IDataOptions = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            {
                name: "[Core Product Group]",
                isNamedSet: true
            },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' },
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Configuring authentication

Users can configure basic authentication information to access the OLAP cube using the [authentication](#) property. The settings required to configure are as follows:

- [userName](#): It allows the user to set a username that recognizes the basic authentication of the IIS.

- [password](#): It allows to set the appropriate password.

If the user does not configure the authentication, a default popup will appear in the browser to get the authentication information.

```
`ts
```

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';

import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';

function App() {
  let dataSourceSettings: IDataOptions = {
    catalog: 'Adventure Works DW 2008 SE',
    cube: 'Adventure Works',
    providerType: 'SSAS',
    enableSorting: true,
    url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
    localeIdentifier: 1033,
    rows: [
      { name: '[Customer].[Customer Geography]', caption: 'Customer Geography' },
    ],
    columns: [
      { name: '[Product].[Product Categories]', caption: 'Product Categories' },
      { name: '[Measures]', caption: 'Measures' },
    ],
    values: [
      { name: '[Measures].[Customer Count]', caption: 'Customer Count' },
      { name: '[Measures].[Internet Sales Amount]', caption: 'Internet Sales Amount' }
    ],
    filters: [
      { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
    ],
    filterSettings: [
      {
        name: '[Date].[Fiscal]', items: ['[Date].[Fiscal].[Fiscal Quarter].[2002]&[4]',
```

```

'[/Date].[/Fiscal].[/Fiscal Year].&[2005]'],
levelCount: 3
}
],
authentication: {
  userName: 'username',
  password: 'password'
}
};
let pivotObj: PivotViewComponent;
return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings} showFieldList={true}><Inject services={{FieldList}} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
`

```

Roles

SQL Server Analysis Services uses [roles](#) to limit data access within a cube. Each role defines a set of permissions that can be granted to a single user or groups of users. It is used to manage security by limiting access to sensitive data and determining who has access to and can change the cube. It can be configured using the [roles](#) property in [dataSourceSettings](#).

The [roles](#) property can be used to specify one or more roles to the OLAP cube, separated by commas.

```

`ts
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { IDataOptions, IDataset, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
function App() {
  let dataSourceSettings: IDataOptions = {
    catalog: 'Adventure Works DW 2008 SE',
    cube: 'Adventure Works',
    providerType: 'SSAS',
    roles: 'Role1',
    url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
    localeIdentifier: 1033,

```

```

rows: [
{ name: '[Customer].[Customer Geography]', caption: 'Customer Geography' },
],
columns: [
{ name: '[Product].[Product Categories]', caption: 'Product Categories' },
{ name: '[Measures]', caption: 'Measures' },
],
values: [
{ name: '[Measures].[Customer Count]', caption: 'Customer Count' },
{ name: '[Measures].[Internet Sales Amount]', caption: 'Internet Sales Amount' }
]
};

let pivotObj: PivotViewComponent;

return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}><Inject services={[]} />
</PivotViewComponent>);

};

export default App;

ReactDOM.render(<App />, document.getElementById('sample'));
`

```

OLAP Cube: Elements

Field list

The field list, aka cube dimension browser, is a tree view like structure that organizes the cube elements such as dimensions, hierarchies, measures, etc., from the selected cube into independent logical groups.

Types of node in field list

- **Display folder:** A folder that contains a set of similar elements.
- **Measure:** Quantity available for analysis.
- **Dimension:** A name given to the parts of the cube that categorizes data.
- **Attribute Hierarchy:** Level of attributes down the hierarchy.
- **User-defined Hierarchy:** Members of a dimension in a hierarchical structure.
- **Level:** Denotes a specific level in the category.
- **Named Set:** A collection of tuples and members, that can be defined and saved as a part of cube definition for later use.

Measure

In a cube, a measure is a set of values that are based on a column in the cube's fact table and are usually numeric. The measures are the central values of a cube that are analyzed. That is, measures are the

numeric data of primary interest to users browsing a cube. You can select measures depend on the types of users request. Some common measures are sales, costs, expenditures, and production count.

Dimension

A simple dimension object is composed of basic information such as name, hierarchy, level, and members. You can create a dimension element by specifying its name and providing the hierarchy and level name. The dimension element contains the hierarchical details and information about each included level elements in that hierarchy. A hierarchy can have any number of level elements and the level elements can have any number of members and the member elements can have any number of child members.

Hierarchy

Each element of a dimension can be summarized using a hierarchy. The hierarchy is a series of parent-child relationship, where a parent member represents the consolidation of members which are its children. Parent members can be further aggregated as the children of another parent. For example, May 2005 can be summarized into Second Quarter 2005 which in turn would be summarized in the year 2005.

Level

Level element is the child of hierarchy element which contains a set of members, each of which has the same rank within a hierarchy.

Attribute hierarchy

Attribute hierarchy contains the following levels:

- A leaf level contains distinct attribute member, and each member of the leaf level is known as a leaf member.
- Intermediate levels if the attribute hierarchy is a parent-child hierarchy.
- An optional (all) level contains the aggregated value of the attribute hierarchy's leaf members, with the member of the (all) level also known as the (all) member.

User-defined hierarchy

User-defined hierarchy organizes the members of a dimension into hierarchical structure and provides navigation paths in a cube. For example, take a dimension table that supports three attributes such as year, quarter, and month. The year, quarter, and month attributes are used to construct a user-defined hierarchy, named Calendar, in the time dimension that relates to all levels.

Differentiating user-defined hierarchy and attribute hierarchy

- User-defined hierarchy contains more than one level whereas attribute hierarchy contains only one level.
- User-defined hierarchy provides the navigation path between the levels taken from attribute hierarchies of the same dimension.
- The attribute hierarchy and the user-defined hierarchy are represented in different ways as shown in the following table.

Named set

A named set is a collection of tuples and members, which can be defined and saved as a part of the cube definition. Named set records reside inside the sets folder, which is under a dimension element. These elements can be dragged to [rows](#) or [columns](#) axis via grouping bar or field list at runtime. To work with a

lengthy, complex, or commonly used expression easier, Multidimensional Expressions (MDX) allows you to define a named set.

Calculated field

The calculated field allows user to insert or add a new calculated field based on the available OLAP cube elements from the bound data source. Calculated fields are nothing but customized dimensions or measures that are newly created based on the user-defined expression.

The two types of calculated fields are as follows:

- **Calculated Measure** – Creates a new measure through user-defined expression.
- **Calculated Dimension** – Creates a new dimension through user-defined expression.

Symbolic representation of the nodes inside field list

| Icon | Name | Node type | Is Draggable |

|-----|-----|-----|-----|



|
False|

| Display Folder | Display Folder |



|

| Measure | Measure | False |



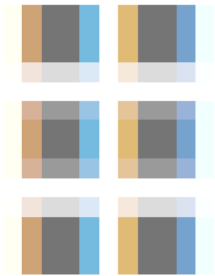
|

| Dimension | Dimension | False |



|

| User Defined Hierarchy| Hierarchy| True|



|

| Attribute Hierarchy| Hierarchy| True|



|





Element| True|

| Levels (in order)| Level



| | Named Set| Named Set| True|

In general, the Pivot Table is created using the built-in engine for given data source. This is an optional feature that allows you to create the Pivot Table with a server-side pivot engine and external data binding. And this option is applicable only for relational data source.

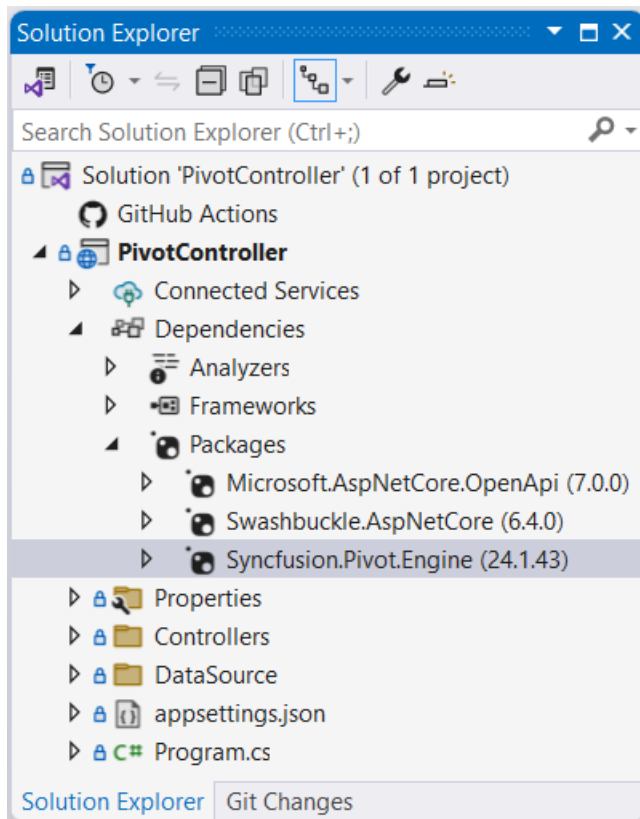
Server side pivot engine in React Pivotview component

This section briefs the Syncfusion assembly [Syncfusion.Pivot.Engine](#), which is used in a server-side application to perform all Pivot calculations such as aggregation, filtering, sorting, grouping, and so on, and only the information to be displayed in the Pivot Table's viewport is passed to the client-side (browser) via web service (Web API) rather than the entire data source. It reduces network traffic and improves the rendering performance of the Pivot Table, especially when dealing with large amounts of data. It also works best with virtual scrolling enabled and supports all the Pivot Table's existing features.

Quick steps to render the Pivot Table by using the server-side Pivot Engine

Download and installing Server-side Pivot Engine

1. Download the ASP.NET Core-based stand-alone Pivot Table [application](#) from the GitHub repository.
2. The **PivotController** (Server-side) application that is downloaded includes the following files.
 - **PivotController.cs** file under **Controllers** folder – This helps to do data communication with Pivot Table.
 - **DataSource.cs** file under **DataSource** folder – This file has model classes to define the structure of the data sources.
 - The sample data source files **sales.csv** and **sales-analysis.json** under **DataSource** folder.
3. Open the **PivotController** application in Visual Studio where the Syncfusion library [Syncfusion.Pivot.Engine](#) will be downloaded automatically from the nuget.org site.



Connecting Pivot Table to Server-side Pivot Engine

1. Run the **PivotController** (Server-side) application which will be hosted in IIS shortly.

2. Then in the Pivot Table sample, set the [mode](#) property under [dataSourceSettings](#) as **Server** and map the URL of the hosted Server-side application in [URL](#) property of [dataSourceSettings](#).

```
`javascript
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let dataSourceSettings: IDataOptions = {
    url: 'https://localhost:44350/api/pivot/post',
    mode: 'Server',
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
    dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

3. Frame and set the report based on the data source available in the **PivotController** application.

```
`javascript
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let dataSourceSettings: IDataOptions = {
    url: 'https://localhost:44350/api/pivot/post',
    mode: 'Server',
    rows: [{
      name: 'ProductID', caption: 'Product ID'
    }],
    formatSettings: [{
      name: 'Price', format: 'C'
    }],
    columns: [{
```

```

name: 'Year', caption: 'Production Year'
}},
values: [
{ name: 'Sold', caption: 'Units Sold' },
{ name: 'Price', caption: 'Sold Amount' }
],
}
let pivotObj: PivotViewComponent;
return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
`

```

4. Run the sample to get the following result.

	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
PRO-10	7901220	\$1,841,712.00	7561880	\$1,762,804.00	7591220
PRO-11	7174625	\$1,673,533.00	7338175	\$1,711,599.80	7164625
PRO-12	9168140	\$2,135,848.80	7689400	\$1,792,704.00	7375140
PRO-13	6674280	\$1,560,619.20	8585260	\$2,001,738.40	6574280
PRO-14	7489240	\$1,745,841.20	7167840	\$1,675,479.20	7396240
PRO-15	8149185	\$1,902,397.00	7450585	\$1,734,093.00	7249185
PRO-16	7070150	\$1,652,838.40	8037500	\$1,873,597.60	7880150

Available configurations in Server-side application

[Supportive Data Sources](#)

The server-side Pivot Engine supports the following data sources,

- Collection
- JSON
- CSV
- DataTable
- Dynamic

[Collection](#)

The collection data sources such as List, IEnumerable, and so on are supported. This can be bound using the **GetData** controller method. Also, in the Pivot Table sample, set the [type](#) property under [dataSourceSettings](#) to **JSON**, which is also the default enumeration value.

In the server-side application (**PivotController**), a collection type data source is framed in the **DataSource.cs** file as shown in the following.

```
`csharp
public class PivotViewData
{
    public string ProductID { get; set; }
    public string Country { get; set; }
    public string Product { get; set; }
    public double Sold { get; set; }
    public double Price { get; set; }
    public string Year { get; set; }
    public List<PivotViewData> GetVirtualData()
    {
        List<PivotViewData> VirtualData = new List<PivotViewData>();
        for (int i = 1; i <= 10000; i++)
        {
            PivotViewData p = new PivotViewData
            {
                ProductID = "PRO-" + ((100 + i)%20),
                Year = (new string[] { "FY 2015", "FY 2016", "FY 2017", "FY 2018", "FY 2019" })[new Random().Next(5)],
                Country = (new string[] { "Canada", "France", "Australia", "Germany", "France" })[new
                Random().Next(5)],
                Product = (new string[] { "Car", "Van", "Bike", "Flight", "Bus" })[new Random().Next(5)],
                Price = (3.4 * i) + 500,
                Sold = (i * 15) + 10
            };
            VirtualData.Add(p);
        }
        return VirtualData;
    }
}
```

To bind the data source, set its model type **PivotViewData** to **TValue** of the **PivotEngine** class.

```
`csharp
```

```
private PivotEngine<DataSource.PivotViewData> PivotEngine = new
PivotEngine<DataSource.PivotViewData>();
`
```

Then call the data source in **GetData** method of **PivotController.cs** file.

```
`csharp
public async Task<object> GetData(FetchData param)
{
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
    async (cacheEntry) =>
    {
        cacheEntry.SetSize(1);
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        // Here bind the collection type data source.
        return new DataSource.PivotViewData().GetVirtualData();
    });
}
`
```

Finally set the appropriate report to the Pivot Table sample based on the above data source.

```
`javascript
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let dataSourceSettings: IDataOptions = {
        url: 'https://localhost:44350/api/pivot/post',
        mode: 'Server',
        rows: [{
            name: 'ProductID', caption: 'Product ID'
        }],
        formatSettings: [{
            name: 'Price', format: 'C'
        }],
        columns: [{
```

```

name: 'Year', caption: 'Production Year'
}},
values: [
{ name: 'Sold', caption: 'Units Sold' },
{ name: 'Price', caption: 'Sold Amount' }
],
}
let pivotObj: PivotViewComponent;
return <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
`

```

	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
PRO-10	7901220	\$1,841,712.00	7561880	\$1,762,804.00	7591220
PRO-11	7174625	\$1,673,533.00	7338175	\$1,711,599.80	7164625
PRO-12	9168140	\$2,135,848.80	7689400	\$1,792,704.00	7375140
PRO-13	6674280	\$1,560,619.20	8585260	\$2,001,738.40	6574280
PRO-14	7489240	\$1,745,841.20	7167840	\$1,675,479.20	7392240
PRO-15	8149185	\$1,902,397.00	7450585	\$1,734,093.00	7249185
PRO-16	7070150	\$1,652,838.40	8037500	\$1,873,597.60	7880150

JSON

The JSON data from a local *.json file type can be connected to the Pivot Table. Here, the file can be read by the **StreamReader** option, which will give the result in the string format. The resultant string needs to be converted to collect data that can be bound to the Server-side pivot engine.

In the Server-side application, **sales-analysis.json** file is available under **DataSource** folder and its model type is defined in **DataSource.cs** file.

```

`csharp
public class PivotJSONData
{
    public string Date { get; set; }
    public string Sector { get; set; }
    public string EnerType { get; set; }
    public string EneSource { get; set; }
}

```

```

public int PowUnits { get; set; }
public int ProCost { get; set; }
public List<PivotJSONData> ReadJSONData(string url)
{
    WebClient myWebClient = new WebClient();
    Stream myStream = myWebClient.OpenRead(url);
    StreamReader stream = new StreamReader(myStream);
    string result = stream.ReadToEnd();
    stream.Close();
    return Newtonsoft.Json.JsonConvert.DeserializeObject<List<PivotJSONData>>(result);
}
}
`

```

To bind the data source, set its model type **PivotJSONData** to **TValue** of the **PivotEngine** class.

```

`csharp
private PivotEngine<DataSource.PivotJSONData> PivotEngine = new PivotEngine<DataSource.
PivotJSONData>();
`

```

Then call the data source in **GetData** method of **PivotController.cs** file.

```

`csharp
public async Task<object> GetData(FetchData param)
{
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
    async (cacheEntry) =>
    {
        cacheEntry.SetSize(1);
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        // Here bind JSON type data source from the sales-analysis.json file.
        return new DataSource.PivotJSONData().ReadJSONData(_hostingEnvironment.ContentRootPath +
        "//DataSource//sales-analysis.json");
    });
}
`

```

Finally set the appropriate report to the Pivot Table sample based on the above data source.

```
`javascript
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let dataSourceSettings: IDataOptions = {
    url: 'https://localhost:44350/api/pivot/post',
    mode: 'Server',
    type: 'JSON',
    rows: [{
      name: 'EneSource', caption: 'Energy Source'
    }],
    formatSettings: [{
      name: 'ProCost', format: 'C'
    }],
    columns: [{
      name: 'EnerType', caption: 'Energy Type'
    }],
    values: [
      { name: 'PowUnits', caption: 'Units Sold' },
      { name: 'ProCost', caption: 'Sold Amount' }
    ],
  };
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
    dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

	Biomass		Free Energy		Grand Total
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
Bio-diesel	1042	\$1,439.00			1042
Ethanol Fuel	595	\$1,031.00			595
Geo-thermal			1528	\$2,115.00	1528
Hydro-electric			3378	\$3,244.00	3378
Solar			7929	\$6,210.00	7929
Wastage	712	\$1,043.00			712
Wind			15571	\$7,666.00	15571

JSON data from any remote server, like a local JSON file, can also be supported. It accepts both directly downloadable files (*.json*) and *web service URLs*. To bind this, the URL of the .json file of a remote server has to be mapped under the **GetData** method. The rest of the configurations are the same as described above.

In the server-side application, the CDN link is used to connect the same **sales-analysis.json** file which is already hosted in the Syncfusion server.

```
`csharp
```

```
public async Task<object> GetData(FetchData param)
{
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
    async (cacheEntry) =>
    {
        cacheEntry.SetSize(1);
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        // Here bind JSON type data source from remote server.
        return new DataSource.PivotJSONData().ReadJSONData("http://cdn.syncfusion.com/data/sales-
        analysis.json");
    });
}
```

CSV

The CSV data from a local *.csv file type can be connected to the Pivot Table. Here, the file can be read by the **StreamReader** option, which will give the result in the string format. The resultant string needs to be converted to collect data that can be bound to the server-side pivot engine. Also, in the Pivot Table sample, set the **type** property under [dataSourceSettings](#) as **CSV**.

In the server application, the **sales.csv** file is available under the **DataSource** folder, and its model type is defined in the **DataSource.cs** file.

```
`csharp
```

```
public class PivotCSVData
{
    public string Region { get; set; }
    public string Country { get; set; }
    public string ItemType { get; set; }
    public string SalesChannel { get; set; }
    public string OrderPriority { get; set; }
    public string OrderDate { get; set; }
    public int OrderID { get; set; }
    public string ShipDate { get; set; }
    public int UnitsSold { get; set; }
    public double UnitPrice { get; set; }
    public double UnitCost { get; set; }
    public double TotalRevenue { get; set; }
    public double TotalCost { get; set; }
    public double TotalProfit { get; set; }
    public List<string[]> ReadCSVData(string url)
    {
        List<string[]> data = new List<string[]>();
        using (StreamReader reader = new StreamReader(new WebClient().OpenRead(url)))
        {
            string line;
            while ((line = reader.ReadLine()) != null)
            {
                line = line.Trim();
                if (!string.IsNullOrEmpty(line))
                {
                    data.Add(line.Split(','));
                }
            }
            return data;
        }
    }
}
```

```
}
,
```

To bind the data source, set its model type **PivotCSVData** to **TValue** of the **PivotEngine** class.

```
`csharp
```

```
private PivotEngine<DataSource.PivotCSVData> PivotEngine = new PivotEngine<DataSource.
PivotCSVData>();
```

```
,
```

Then call the data source in **GetData** method of **PivotController.cs** file.

```
`csharp
```

```
public async Task<object> GetData(FetchData param)
```

```
{
```

```
return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
async (cacheEntry) =>
```

```
{
```

```
cacheEntry.SetSize(1);
```

```
cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
```

```
// Here bind CSV type data source from sales.csv file.
```

```
return new DataSource.PivotCSVData().ReadCSVData(_hostingEnvironment.ContentRootPath +
"//DataSource//sales.csv");
```

```
});
```

```
}
```

```
,
```

Finally set the appropriate report to the Pivot Table sample based on the above data source.

```
`javascript
```

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
```

```
import * as React from 'react';
```

```
import * as ReactDOM from 'react-dom';
```

```
function App() {
```

```
let dataSourceSettings: IDataOptions = {
```

```
url: 'https://localhost:44350/api/pivot/post',
```

```
mode: 'Server',
```

```
type: 'CSV',
```

```
rows: [{
```

```
name: 'ItemType', caption: 'Item Type'
```



```

    }},
    formatSettings: [{
      name: 'UnitPrice', format: 'C'
    }],
    columns: [{
      name: 'Region'
    }],
    values: [
      { name: 'UnitsSold', caption: 'Units Sold' },
      { name: 'UnitPrice', caption: 'Sold Amount' }
    ],
  },
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
    dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
`

```

	Asia		Australia and Oceania		Central Ar
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
Baby Food	55810	\$3,318.64	21548	\$765.84	
Beverages	84400	\$806.65	34486	\$332.15	
Cereal	32668	\$1,439.90	43195	\$1,645.60	
Clothes	24290	\$764.96	17020	\$655.68	
Cosmetics	84630	\$7,432.40	35220	\$3,060.40	
Fruits	35679	\$74.64	29951	\$46.65	
Household	28166	\$4,009.62	55444	\$6,014.43	

CSV data from any remote server, like a local CSV file, can also be supported. It accepts both directly downloadable files (.csv) and web service URLs. To bind this, the URL of the .csv file of a remote server has to be mapped under **GetData** method. The rest of the configurations are the same as described above.

In the server application, the CDN link is used to connect the same **sales.csv** file which is already hosted in the Syncfusion server.

```
`csharp
```

```

public async Task<object> GetData(FetchData param)
{
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
    async (cacheEntry) =>
    {
        cacheEntry.SetSize(1);
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        // Here bind CSV type data source from remote server.
        return new DataSource.PivotCSVData().ReadCSVData("http://cdn.syncfusion.com/data/sales-
analysis.csv");
    });
}
`

```

DataTable

In the server-side application, there is a manually created DataTable **BusinessObjectsDataView** by mapping the model type **PivotViewData** in **DataSource.cs** file.

```

`csharp
public class BusinessObjectsDataView
{
    public DataTable GetDataTable()
    {
        DataTable dt = new DataTable("BusinessObjectsDataTable");
        PropertyDescriptorCollection pdc = TypeDescriptor.GetProperties(typeof(PivotViewData));
        foreach (PropertyDescriptor pd in pdc)
        {
            dt.Columns.Add(new DataColumn(pd.Name, pd.PropertyType));
        }
        List<PivotViewData> list = new PivotViewData().GetVirtualData();
        foreach (PivotViewData bo in list)
        {
            DataRow dr = dt.NewRow();
            foreach (PropertyDescriptor pd in pdc)
            {
                dr[pd.Name] = pd.GetValue(bo);
            }
        }
    }
}

```

```

}
dt.Rows.Add(dr);
}
return dt;
}
}
`

```

To bind the data source, set its model type **PivotViewData** to **TValue** of the **PivotEngine** class.

```

`csharp
private PivotEngine<DataSource.PivotViewData> PivotEngine = new
PivotEngine<DataSource.PivotViewData>();
`

```

Then call the data source in **GetData** method of **PivotController.cs** file.

```

`csharp
public async Task<object> GetData(FetchData param)
{
return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
async (cacheEntry) =>
{
cacheEntry.SetSize(1);
cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
// Here bind the DataTable.
return new DataSource.BusinessObjectsDataView().GetDataTable();
});
}
`

```

Finally set the appropriate report to the Pivot Table sample based on the above data source.

```

`javascript
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
let dataSourceSettings: IDataOptions = {

```

```

url: 'https://localhost:44350/api/pivot/post',
mode: 'Server',
rows: [{
  name: 'ProductID', caption: 'Product ID'
}],
formatSettings: [{
  name: 'Price', format: 'C'
}],
columns: [{
  name: 'Year', caption: 'Production Year'
}],
values: [
  { name: 'Sold', caption: 'Units Sold' },
  { name: 'Price', caption: 'Sold Amount' }
],
let pivotObj: PivotViewComponent;
return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
  dataSourceSettings={dataSourceSettings}></PivotViewComponent>
);
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
PRO-10	7901220	\$1,841,712.00	7561880	\$1,762,804.00	7591220
PRO-11	7174625	\$1,673,533.00	7338175	\$1,711,599.80	7164625
PRO-12	9168140	\$2,135,848.80	7689400	\$1,792,704.00	7378140
PRO-13	6674280	\$1,560,619.20	8585260	\$2,001,738.40	6574280
PRO-14	7489240	\$1,745,841.20	7167840	\$1,675,479.20	7392400
PRO-15	8149185	\$1,902,397.00	7450585	\$1,734,093.00	7249185
PRO-16	7070150	\$1,652,838.40	8037500	\$1,873,597.60	7881500

Dynamic

The model type has to be defined in the aforementioned data sources. However, there is no need to define a model type for the following data sources, which are also supported by the server-side pivot engine.

ExpandoObject

In the server-side application, an **ExpandoObject** type data source is available under the class **PivotExpandoData** in **DataSource.cs** file.

```
`csharp
public class PivotExpandoData
{
    public List<ExpandoObject> Orders { get; set; } = new List<ExpandoObject>();
    public List<ExpandoObject> GetExpandoData()
    {
        Orders = Enumerable.Range(1, 75).Select((x) =>
        {
            dynamic d = new ExpandoObject();
            d.OrderID = 1000 + (x % 100);
            d.CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID" })[new
            Random().Next(5)];
            d.Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x;
            d.OrderDate = (new DateTime[] { new DateTime(2010, 11, 5), new DateTime(2018, 10, 3), new
            DateTime(1995, 9, 9), new DateTime(2012, 8, 2), new DateTime(2015, 4, 11) })[new Random().Next(5)];
            d.ShipCountry = (new string[] { "USA", "UK" })[new Random().Next(2)];
            d.Verified = (new bool[] { true, false })[new Random().Next(2)];
            return d;
        }).Cast<ExpandoObject>().ToList<ExpandoObject>();
        return Orders;
    }
}
```

To bind the data source, set its model type as **ExpandoObject** to **TValue** of the **PivotEngine** class.

```
`csharp
private PivotEngine<ExpandoObject> PivotEngine = new PivotEngine<ExpandoObject>();
`
```

Then call the data source in **GetData** method of **PivotController.cs** file.

```
`csharp
public async Task<object> GetData(FetchData param)
{
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
    async (cacheEntry) =>
    {
        cacheEntry.SetSize(1);
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        // Here returns ExpandoObject type data source.
        return new DataSource.PivotExpandoData().GetExpandoData();
    });
}
```

Finally set the appropriate report to the Pivot Table sample based on the above data source.

```
`javascript
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

function App() {
    let dataSourceSettings: IDataOptions = {
        url: 'https://localhost:44350/api/pivot/post',
        mode: 'Server',
        rows: [{
            name: 'CustomerID', caption: 'Customer ID'
        }],
        columns: [{
            name: 'ShipCountry', caption: 'Ship Country'
        }],
        values: [
            { name: 'Freight', caption: 'Units Sold' }
        ],
    }

    let pivotObj: PivotViewComponent;
```

```

return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
`

```

	UK	USA	Grand Total
	Units Sold	Units Sold	Units Sold
ALFKI	1187	738	1925
ANANTR	1011	814	1825
ANTON	1046	632	1678
BLONP	1080	749	1829
BOLID	492	929	1421
Grand Total	4816	3862	8678

Dynamic Objects

In the server-side application, a data source is framed by dynamic objects which is available under the class **PivotDynamicData** in the **DataSource.cs** file.

```

`csharp
public class PivotDynamicData
{
    public List<DynamicDictionary> Orders = new List<DynamicDictionary>() { };
    public List<DynamicDictionary> GetDynamicData()
    {
        Orders = Enumerable.Range(1, 100).Select((x) =>
        {
            dynamic d = new DynamicDictionary();
            d.OrderID = 100 + x;
            d.CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID" })[new
            Random().Next(5)];
            d.Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x;
            d.OrderDate = (new DateTime[] { new DateTime(2010, 11, 5), new DateTime(2018, 10, 3), new
            DateTime(1995, 9, 9), new DateTime(2012, 8, 2), new DateTime(2015, 4, 11) })[new Random().Next(5)];
            d.ShipCountry = (new string[] { "USA", "UK" })[new Random().Next(2)];
            d.Verified = (new bool[] { true, false })[new Random().Next(2)];
            return d;
        }).Cast<DynamicDictionary>().ToList<DynamicDictionary>();
    }
}

```

```

return Orders;
}
public class DynamicDictionary : System.Dynamic.DynamicObject
{
    Dictionary<string, object> dictionary = new Dictionary<string, object>();
    public override bool TryGetMember(GetMemberBinder binder, out object result)
    {
        string name = binder.Name;
        return dictionary.TryGetValue(name, out result);
    }
    public override bool TrySetMember(SetMemberBinder binder, object value)
    {
        dictionary[binder.Name] = value;
        return true;
    }
    //The "GetDynamicMemberNames" method of the "DynamicDictionary" class must be overridden and
    return the property names to perform data operation and editing while using dynamic objects.
    public override System.Collections.Generic.IEnumerable<string> GetDynamicMemberNames()
    {
        return this.dictionary?.Keys;
    }
}

```

To bind the data source, set its class **PivotDynamicData** to **TValue** of the **PivotEngine** class.

```

`csharp
private PivotEngine<DataSource.PivotDynamicData> PivotEngine = new
PivotEngine<DataSource.PivotDynamicData>();

```

Then call the data source in **GetData** method of **PivotController.cs** file.

```

`csharp
public async Task<object> GetData(FetchData param)
{
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,

```



```

async (cacheEntry) =>
{
  cacheEntry.SetSize(1);
  cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
  // Here bind data source with dynamic objects.
  return new DataSource.PivotDynamicData().GetDynamicData();
});
}
`

```

Finally set the appropriate report to the Pivot Table sample based on the above data source.

```

`javascript
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let dataSourceSettings: IDataOptions = {
    url: 'https://localhost:44350/api/pivot/post',
    mode: 'Server',
    rows: [{
      name: 'CustomerID', caption: 'Customer ID'
    }],
    columns: [{
      name: 'ShipCountry', caption: 'Ship Country'
    }],
    values: [
      { name: 'Freight', caption: 'Units Sold' }
    ],
  };
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
    dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;

```

```
ReactDOM.render(<App />, document.getElementById('sample'));
```

```
,
```

	UK	USA	Grand Total
	Units Sold	Units Sold	Units Sold
ALFKI	1030	742	1772
ANANTR	1352	1018	2370
ANTON	2525	1782	4307
BLONP	1786	1002	2788
BOLID	1409	2012	3421
Grand Total	8102	6556	14658

Controller Configuration

Memory Cache

In the server-side application, the [Memory Cache](#) option is used to store the data source and engine properties in RAM, which will be used for UI operations. To improve performance, this limits the execution of all initial rendering code to regenerate the aggregated values during each UI operation. The codes below show how we use the memory cache option in the **GetEngine** method to store engine properties.

```
`csharp
public async Task<EngineProperties> GetEngine(FetchData param)
{
    isRendered = false;
    // Engine properties are stored in memory cache with GUID "param.Hash".
    return await _cache.GetOrCreateAsync("engine" + param.Hash,
    async (cacheEntry) =>
    {
        isRendered = true;
        cacheEntry.SetSize(1);
        // Memory cache expiration time can be set here.
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        PivotEngine.Data = await GetData(param);
        return await PivotEngine.GetEngine(param);
    });
}
```

The engine properties are stored in RAM as a cache with a unique ID (GUID) that is transferred from the client-side source code. The GUID is generated at random and will be changed if the page containing the

Pivot Table is refreshed or opened in a new tab/window. As a result, each GUID's memory cache contains unique information, and the component operates independently.

Meanwhile, the memory cache is set to expire after 60 minutes from RAM to free its memory. If the component is still running, the data will be generated and stored for another 60 minutes.

Methods and its needs

- **Post:** Allows to get the information from the client-side source and calls appropriate controller methods.
- **GetEngine:** Allows to store the engine properties in RAM as a cache which fires on initial rendering or when the memory cache is expired.
- **GetData:** Allows to store data source in RAM as a cache which fires on initial rendering or when the memory cache is expired.
- **GetMembers:** Allows to get the members of a field. This fires when the member editor is opened to do a filtering operation.
- **GetRawData:** Allows to get raw data of an aggregated value cell. This fires when the drill-through or editing dialog is opened.
- **GetPivotValues:** Allows to update the stored engine properties in-memory cache and returns the aggregated values to browser to render the Pivot Table. Here, the return value can be modified. The Pivot Table will be rendered browser-based on this.

Pivot chart in React Pivotview component

In pivot table component, pivot chart would act as an additional visualization component with its basic and important characteristic like drill down and drill up, 15+ chart types, series customization, axis customization, legend customization, export, print and tooltip. Its main purpose is to show the pivot data in graphical format.

If user prefers, the pivot chart component can also be displayed individually with pivot values and can change the report dynamically with the help of field list and grouping bar. Using the [displayOption](#) property, user can set the visibility of grid and chart in pivot table component. It holds below properties,

- [view](#): Specifies the pivot table component to display grid alone or chart alone or both.
- [primary](#): Specifies the pivot table to display either grid or chart as primary component during initial loading. It is applicable only when setting the property [view](#) to **Both**.

You should inject the PivotChart module to make the its features available in the pivot table.

The below sample displays the pivot chart component based on the pivot report bound on it.

INDEX.JSX

```
import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
```

```

        chartSeries: { type: 'Column' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
        chartSeries: { type: 'Column' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}/></PivotViewComponent>);
};

```

```
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Data Binding

End user can bind both local and remote data binding options available in the component to feed the data. The [dataSource](#) property can be assigned either with an instance of [DataManager](#) or JavaScript object array collection.

For more information [refer](#) here.

Chart Types

Supports 21 different types of charts as follows,

- Line
- Column
- Area
- Bar
- StepArea
- StackingLine
- StackingColumn
- StackingArea
- StackingBar
- StepLine
- Pareto
- Bubble
- Scatter
- Spline
- SplineArea
- StackingLine100
- StackingColumn100
- StackingBar100
- StackingArea100
- Polar
- Radar

Line is the default pivot chart type. User can change the pivot chart type by using the property [type](#) in [chartSeries](#) class.

In the below code sample, the pivot chart type is set as **Bar**.

INDEX.JSX

```
import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
};
```

```

    let chartSettings = {
      chartSeries: { type: 'Bar' }
    };
    let dataSourceSettings = {
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      dataSource: pivotData,
      expandAll: false,
      filters: [],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
  }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    chartSeries: { type: 'Bar' }
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}/></PivotViewComponent>);

```

```
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Accumulation Charts

Supports 4 different types of accumulation charts as follows,

- Pie
- Doughnut
- Funnel
- Pyramid

As like other chart types it can be changed using the property [type](#) in [chartSeries](#).

In the below code sample, the **Pie** chart is rendered, and the other accumulation charts can be switched using the drop-down list.

INDEX.JSX

```
{% raw %}
import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    chartSeries: { type: 'Pie' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let fields = { text: 'text', value: 'value' };
  let chartTypes = [
    { 'value': 'Pie', 'text': 'Pie' },
    { 'value': 'Doughnut', 'text': 'Doughnut' },
    { 'value': 'Funnel', 'text': 'Funnel' },
    { 'value': 'Pyramid', 'text': 'Pyramid' },
  ];
  let pivotObj;
  function ddlOnChange(args) {
    pivotObj.chartSettings.chartSeries.type = args.value;
  }
}
```

```

    return (<div><div className="container" style={{ height: '500px' }}><div
    id="dropdown-control" style={{ marginBottom: '5px' }}><table style={{ width:
    '350px', marginLeft: '50px' }}><tbody><tr style={{ height: '50px'
    }}><td><div><b>Accumulation
    Chart:</b></div></td><td><div><DropDownListComponent floatLabelType={'Auto'}
    fields={fields} change={ddlOnChange.bind(this)} id="charttypes" index={0}
    enabled={true}
    dataSource={chartTypes}></div></td></tr></tbody></table></div><div><PivotVi
    ewComponent height={350} ref={d => pivotObj = d} id='PivotView'
    chartSettings={chartSettings} displayOption={displayOption}
    dataSourceSettings={dataSourceSettings}><Inject
    services={[PivotChart]}></PivotViewComponent></div></div></div>);
  }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    chartSeries: { type: 'Pie' }
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let fields: object = { text: 'text', value: 'value' };
  let chartTypes = [
    { 'value': 'Pie', 'text': 'Pie' },
    { 'value': 'Doughnut', 'text': 'Doughnut' },
    { 'value': 'Funnel', 'text': 'Funnel' },
    { 'value': 'Pyramid', 'text': 'Pyramid' },
  ];
  let pivotObj: PivotViewComponent;
  function ddlOnChange(args) {

```



```

        pivotObj.chartSettings.chartSeries.type = args.value;
    }
    return (<div><div className="container" style={{ height: '500px' }}><div
id="dropdown-control" style={{marginBottom: '5px'}}><table style={{width:
'350px', marginLeft: '50px'}}><tbody><tr style={{ height: '50px' }}
><td><div><b>Accumulation
Chart:</b></div></td><td><div><DropDownListComponent
floatLabelType={'Auto'} fields={fields} change={ddlOnChange.bind(this)}
id="charttypes" index={0} enabled={true}
dataSource={chartTypes}></div></td></tr></tbody></table></div><div><PivotVi
ewComponent height={350} ref={d => pivotObj = d} id='PivotView'
chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}></PivotViewComponent></div></div></div>;
    );
    export default App;
    ReactDOM.render(<App />, document.getElementById('sample'));
    {% endraw %}

```

Drill Down/Up

In the accumulation charts, drill down and drill up operations can be performed using the built-in context menu option. It will be shown while clicking on the chart series. The context menu has the following options:

Expand - It is to drill down the corresponding series until the last level.

Collapse - It is to drill up the corresponding series until the first level.

Exit - It is to close the context menu.

The drill operation in accumulation charts can be performed only for row headers.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        chartSeries: { type: 'Pie' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Products' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Year', caption: 'Production
Year' }, { name: 'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' } ]
    };
}

```

```

    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

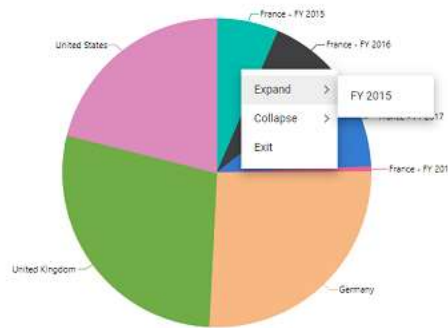
```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
        chartSeries: { type: 'Pie' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Products' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Year', caption: 'Production
Year' }, { name: 'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```



Column Headers and Delimiters

Unlike other chart types, the accumulation charts consider the values of a single column from the pivot table to be drawn. Preferably the first column of the pivot table is considered by default. But it can be changed by defining the column headers using the `columnHeader` property in [chartSettings](#).

If the column has more than one header, then need to mention all the headers separated by the delimiter -, for example, **Germany-Road Bikes**. Using the property `columnDelimiter` in [chartSettings](#), one can set the desired delimiter to separate the column headers.

INDEX.JSX

```
import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        columnHeader: "Germany-Road Bikes", columnDelimiter: "-",
        chartSeries: { type: 'Doughnut' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Country' }, { name: 'Products' }],
        dataSource: pivotData,
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['Germany'] }],
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Year' }, { name: 'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
}
;
```

```
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    columnHeader: "Germany-Road Bikes", columnHeader: "-",
    chartSeries: { type: 'Doughnut' }
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Country' }, { name: 'Products' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    drilledMembers: [{ name: 'Country', items: ['Germany'] }],
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Year' }, { name: 'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject services={[PivotChart]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Label Customization

The data labels are visible by default showing header name. Its visibility can be modified using the `visible` boolean property in `dataLabel`. With regard to the label arrangement, the **Smart Labels** options help to arrange labels efficiently without overlapping. It can be disabled by setting the `enableSmartLabels` property in `chartSettings` as `false`.

The `position` property in `dataLabel` allows to specify the position of the data label. The available options are,

- **Outside**: Positions the label outside the point. It is the default option.
- **Inside**: Positions the label inside the point.

In the following code sample, the data labels are placed inside.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        enableSmartLabels: false,
        chartSeries: { dataLabel: { visible: true, position: "Inside" },
type: 'Pyramid' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' } ]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
        enableSmartLabels: false,
        chartSeries: { dataLabel: {visible:true, position: "Inside" }, type:
'Pyramid' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {

```

```

        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject services={[PivotChart]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

The **Connector Line** will be visible when the data label is placed outside the chart. It can be customized using the **connectorStyle** property in **dataLabel** for its color, length, width etc. In the following code sample, the connector line is customized.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        chartSeries: {
            dataLabel: { visible: true, position: 'Outside', connectorStyle:
{ length: '50px', width: 2, dashArray: '5,3', color: '#f4429e' } },
            type: 'Funnel'
        }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}

```

```

dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
        chartSeries: {
            dataLabel: { visible: true, position: 'Outside', connectorStyle: {
length: '50px', width: 2, dashArray: '5,3', color: '#f4429e' } },
            type: 'Funnel'
        }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject services={[PivotChart]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Pie and Doughnut Customization

User can draw pie and doughnut charts within the specified range using the `startAngle` and `endAngle` properties in [chartSeries](#). The default value of the `startAngle` property is `0`, and the `endAngle` property is `360`. By customizing these properties, user can draw semi pie and semi doughnut charts.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        chartSeries: { startAngle: 270, endAngle: 90, type: 'Doughnut' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
        chartSeries: { startAngle: 270, endAngle: 90, type: 'Doughnut' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],

```



```

        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject services={[PivotChart]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Users can get doughnut chart from pie chart and vice-versa using the `innerRadius` property in [chartSeries](#). If the property is greater than 0 percent, the doughnut chart will appear from the pie chart.

It takes the value only in percentage.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        chartSeries: { innerRadius: "140", type: 'Pie' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataSet, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    chartSeries: { innerRadius: "140", type: 'Pie' }
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataSet[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Exploding Series Points

Exploding can be enabled by setting the `explode` property in `chartSeries` to `true`. The series points will be exploded either on mouse click or touch.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    chartSeries: { explode: true, type: 'Pie' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,

```

```

        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
        chartSeries: { explode: true, type: 'Pie' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject services={[PivotChart]
}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Field List

User can enable the field list by setting the property [showFieldList](#) as **true**.

By using this, user can customize the report dynamically and view the result in pivot chart. For more information regarding the field list, refer the [field list](#) topic.

In the following sample, the Popup mode of field list is enabled in the pivot chart.

INDEX.JSX

```
import { PivotViewComponent, Inject, PivotChart, FieldList } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    chartSeries: { type: 'Bar' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} showFieldList={true}
displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject services={[PivotChart,
FieldList]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart, FieldList } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    chartSeries: { type: 'Bar' }
  } as ChartSettings;
```

```

let dataSourceSettings: IDataOptions = {
  columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
  dataSource: pivotData as IDataset[],
  expandAll: false,
  filters: [],
  formatSettings: [{ name: 'Amount', format: 'C0' }],
  rows: [{ name: 'Country' }, { name: 'Products' }],
  values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
}
let pivotObj: PivotViewComponent;
return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} showFieldList={true}
displayOption={displayOption} dataSourceSettings={dataSourceSettings}
><Inject services={[PivotChart, FieldList]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Grouping Bar

User can enable the grouping bar by setting the property [showGroupingBar](#) as **true**. The grouping bar in pivot chart shows a dropdown list in value axis instead of buttons. The dropdown list holds list of value fields bounded in the [dataSourceSettings](#) and it can be switched to draw the pivot chart with the selected value field. This has been defined as the default behavior in the pivot chart component. For more information regarding the grouping bar, refer the [grouping bar](#) topic.

For multiple axis support, buttons will be placed in value axis instead of dropdown list.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart, GroupingBar } from
'@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    chartSeries: { type: 'Bar' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;

```

```

    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} showGroupingBar={true}
displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject services={[PivotChart,
GroupingBar]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataSet, PivotViewComponent, Inject, DisplayOption,
PivotChart, GroupingBar } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
        chartSeries: { type: 'Bar' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataSet[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} showGroupingBar={true}
displayOption={displayOption} dataSourceSettings={dataSourceSettings}
><Inject services={[PivotChart, GroupingBar]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

For accumulation charts alone, a drop-down list will be placed in the column axis instead of the buttons. The drop-down list shows the column headers available in the pivot table. Users can dynamically switch column headers with the help of the drop-down list, and the accumulation chart will be updated accordingly.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart, GroupingBar } from
'@syncfusion/ej2-react-pivotview';

```

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    chartSeries: { type: 'Pie' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} showGroupingBar={true}
displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject services={[PivotChart,
GroupingBar]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart, GroupingBar } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    chartSeries: { type: 'Pie' }
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],

```

```

        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} showGroupingBar={true}
displayOption={displayOption} dataSourceSettings={dataSourceSettings}
><Inject services={[PivotChart, GroupingBar]} /></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Single Axis

By default, the pivot chart will be drawn with the value field (measure) which is set first in the report under value axis. But, user can change to specific value field using the property [value](#) class.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        chartSeries: { type: 'Column' },
        value: 'Amount'
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]} /></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX


```
import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    chartSeries: { type: 'Column' },
    value: 'Amount'
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Multiple axis

User can draw the pivot chart with multiple value fields by setting the property [enableMultipleAxis](#) in [chartSettings](#) as **true**. In the below code sample, the pivot chart will be drawn with both value fields "Sold" and "Amount" available in the [dataSourceSettings](#).

The multiple axis support is not applicable for the accumulation chart types like pie, doughnut, pyramid, and funnel.

INDEX.JSX

```
import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    enableMultipleAxis: true, chartSeries: { type: 'Column' }
  };
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

```

    };
    let dataSourceSettings = {
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      dataSource: pivotData,
      expandAll: false,
      filters: [],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent>);
  }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    enableMultipleAxis: true, chartSeries: { type: 'Column' }
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}></PivotViewComponent>);
};
export default App;

```

```
ReactDOM.render(<App />, document.getElementById('sample'));
```

If the user binds more value fields, the result will be multiple pivot charts, and each chart will shrink within the parent container height. To avoid this, set the [enableScrollOnMultiAxis](#) property in [chartSettings](#) to **true**. By doing so, each pivot chart will only shrink to a minimal "160px" – "180px" height showing a vertical scrollbar for a clear view.

INDEX.JSX

```
import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        enableScrollOnMultiAxis: true, enableMultipleAxis: true,
        chartSeries: { type: 'Column' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }, { name: 'Products', type: 'Count' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    };
    let chartSettings: ChartSettings = {
        enableScrollOnMultiAxis: true, enableMultipleAxis: true,
        chartSeries: { type: 'Column' }
    };
    let dataSourceSettings: IDataset = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }, { name: 'Products', type: 'Count' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

```

    } as DisplayOption;
    let chartSettings: ChartSettings = {
        enableScrollOnMultiAxis: true, enableMultipleAxis: true, chartSeries: {
type: 'Column' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }, { name: 'Products', type: 'Count' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Meanwhile, there is another way to display multiple values in a chart. In this approach, the series drawn from multiple values are grouped and displayed in a single chart. And, based on the values, multiple Y axis scales will be framed with different ranges. This can be achieved by setting the properties [enableMultipleAxis](#) as **true** and [multipleAxisMode](#) as **Single** in [chartSettings](#).

In the following code sample, the pivot chart can be seen as a single chart with multiple value fields such as **Sold** and **Amount** that are drawn as multiple Y axis.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        enableMultipleAxis: true, multipleAxisMode: 'Single', chartSeries: {
type: 'Column' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],

```

```

        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
        enableMultipleAxis: true, multipleAxisMode : 'Single', chartSeries: {
type: 'Column' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Show member based chart series

When multiple axes are enabled, the user can create each chart series with a unique color palette based on members by setting the [showMemberSeries](#) property in [chartSettings](#) to **true**. As a result, user can

easily identify each member enclosed chart series consistently across different measures in the entire chart area.

Furthermore, with a single click over the legend item, you can show or hide specific chart series visibility based on members across different measures in the entire chart area.

INDEX.JSX

```
import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        enableMultipleAxis: true, value: 'Amount', chartSeries: { type:
'Column', animation: { enable: false } }, showMemberSeries: true
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    let pivotObj;
    return <PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
```

```

    enableMultipleAxis: true, value: 'Amount', chartSeries: { type:
'Column', animation: { enable: false } }, showMemberSeries: true
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  }
  let pivotObj: PivotViewComponent;
  return <PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}/></PivotViewComponent>
</>;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Series customization

User can customize series of the pivot chart using [chartSeries](#) in [chartSettings](#) class. The changes handled in the property will be reflected commonly in all chart series.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    chartSeries: { type: 'Column', enableTooltip: false, border: {
color: '#000', width: 2 } }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
  };
  let pivotObj;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}

```

```

dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    chartSeries: { type: 'Column', enableTooltip: false, border: { color:
'#000', width: 2 } }
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' } ]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

User can also customize the pivot chart series individually using the [chartSeriesCreated](#) event, which occurs after the pivot chart series has been created. You can customize each series individually by iterating them.

In the following sample, the even series are hidden in the pivot chart.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```



```

import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    chartSeries: { type: 'Column' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  function chartSeriesCreated(args) {
    for (let pos = 0; pos < args.series.length; pos++) {
      if (pos % 2 == 0) {
        args.series[pos].visible = false;
      }
    }
  }
  let pivotObj;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}
chartSeriesCreated={chartSeriesCreated}><Inject
services={[PivotChart]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart, ChartSeriesCreatedEventArgs } from '@syncfusion/ej2-react-
pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    chartSeries: { type: 'Column' }
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {

```

```

    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  function chartSeriesCreated(args: ChartSeriesCreatedEventArgs): void {
    for (let pos:number = 0; pos < args.series.length; pos++) {
      if (pos % 2 == 0) {
        args.series[pos].visible = false;
      }
    }
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}
chartSeriesCreated={chartSeriesCreated}><Inject
services={[PivotChart]}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Axis customization

User can customize axis of the pivot chart using [primaryXAxis](#) and [primaryYAxis](#) properties in [chartSettings](#).

Axis customization is not applicable for the accumulation chart types like pie, doughnut, pyramid, and funnel.

In the following sample, title of y-axis and x-axis are customized.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    chartSeries: { type: 'Column' },
    primaryXAxis: { title: 'X axis title' },
    primaryYAxis: { title: 'Y axis title' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,

```

```

        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, DisplayOption, Inject,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
        chartSeries: { type: 'Column' },
        primaryXAxis: { title: 'X axis title' },
        primaryYAxis: { title: 'Y axis title' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

One can also customize multi-level labels of primary x-axis by using the [multiLevelLabelRender](#) event in the [chartSettings](#), which fires on rendering each multi-level label in the pivot chart. It has the following parameters:

axis - It holds the information of the current axis.

text - It allows to change the text of the multi-level label.

textStyle - It allows to customize the text font.

alignment - It allows to set the text alignment.

INDEX.JSX

```
import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        chartSeries: { type: 'Column' },
        multiLevelLabelRender(e) {
            e.alignment = 'Near';
            e.textStyle = { fontFamily: 'Bold', fontWeight: '400', size: '16px', color: 'red' };
            if (e.text === ' + United Kingdom') {
                e.text = 'Text Changed';
                e.textStyle = { fontFamily: 'Bold', fontWeight: '800', size: '16px', color: 'Blue' };
            }
        }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d} id='PivotView' chartSettings={chartSettings} displayOption={displayOption} dataSourceSettings={dataSourceSettings}><Inject services={[PivotChart]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, DisplayOption, Inject,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    chartSeries: { type: 'Column' },
    multiLevelLabelRender(e) {
      e.alignment = 'Near';
      e.textStyle = { fontFamily: 'Bold', fontWeight: '400', size: '16px',
color: 'red' };
      if (e.text === ' + United Kingdom') {
        e.text = 'Text Changed';
        e.textStyle = { fontFamily: 'Bold', fontWeight: '800', size: '16px',
color: 'Blue' };
      }
    }
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Legend customization

User can customize legend using [legendSettings](#) in [chartSettings](#). By default, legend will be visible and it can be hidden by setting the property `Visible` in [legendSettings](#) as `false`.

The pivot chart support different types of legend shapes as follows,

- Circle
- Rectangle
- VerticalLine

- Pentagon
- InvertedTriangle
- SeriesType
- Triangle
- Diamond
- Cross
- HorizontalLine

Here **seriesType** would act as the default shape and it can be changed using the property [legendShape](#) in [chartSeries](#).

Also user can set the position of the legend in pivot chart using the property [position](#) in [legendSettings](#). The available options to set the legend position are as follows,

- Auto: Places the legend based on area type. This is the default.
- Top: Displays the legend at the top of the pivot chart.
- Left: Displays the legend at the left of the pivot chart.
- Bottom: Displays the legend at the bottom of the pivot chart.
- Right: Displays the legend at the right of the pivot chart.
- Custom: Displays the legend based on the given x and y values.

By default, the legend is not visible for the accumulation chart types like pie, doughnut, pyramid, and funnel.

INDEX.JSX

```
import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    legendSettings: { position: 'Right' },
    chartSeries: { type: 'Column', legendShape: 'Pentagon' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (
    <PivotViewComponent height={350} ref={d => pivotObj = d}
      id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
    />
  );
}
```

```
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
  let displayOption: DisplayOption = {
    view: 'Chart'
  } as DisplayOption;
  let chartSettings: ChartSettings = {
    legendSettings: { position: 'Right' },
    chartSeries: { type: 'Column', legendShape: 'Pentagon' }
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

User Interaction

Marker and CrossHair

User can enable and customize the marker and crosshair using [markerSettings](#) and [crosshairSettings](#) properties in [chartSettings](#) class respectively.

Also user can enable and customize the crosshair tooltip for axes using [crosshairTooltip](#).

Marker and crosshair is not applicable for the accumulation chart types like pie, doughnut, pyramid, and funnel.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        crosshair: { enable: true },
        chartSeries: {
            type: 'Line',
            marker: { fill: '#EEE', height: 10, width: 10, shape:
'Pentagon', visible: true }
        },
        primaryXAxis: { crosshairTooltip: { enable: true, fill: '#ff0000' } },
    },
    primaryYAxis: { crosshairTooltip: { enable: true, fill: '#0000FF' } }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {

```



```

    crosshair: { enable: true },
    chartSeries: {
      type: 'Line',
      marker: { fill: '#EEE', height: 10, width: 10, shape: 'Pentagon',
visible: true }
    },
    primaryXAxis: { crosshairTooltip: { enable: true, fill: '#ff0000' } },
    primaryYAxis: { crosshairTooltip: { enable: true, fill: '#0000FF' } }
  } as ChartSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataSet[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Zooming and Panning

User can customize zooming and panning option using the property [zoomSettings](#) in [chartSettings](#).

The pivot chart support four types of zooming which can be set as follows,

- [enablePinchZooming](#)
- [enableSelectionZooming](#)
- [enableDeferredZooming](#)
- [enableMouseWheelZooming](#)

and three modes of zooming direction that specifies whether to zoom vertically or horizontally or in both ways which are,

- x: Pivot chart can be zoomed horizontally.
- y: Pivot chart can be zoomed vertically.
- x,y: Pivot chart can be zoomed both vertically and horizontally.

This can be set using the property [mode](#) in [zoomSettings](#). By default, if the pivot chart is zoomed, a toolbar would display with the options - Zoom, ZoomIn, ZoomOut, Pan, Reset. User can also customize its option using the property [toolbarItems](#) in [zoomSettings](#).

Zooming and panning is not applicable for the accumulation chart types like pie, doughnut, pyramid, and funnel.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        chartSeries: {
            type: 'Column'
        },
        zoomSettings: {
            enableDeferredZooming: true,
            enableMouseWheelZooming: true,
            enablePinchZooming: true,
            enableSelectionZooming: true
        }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;

```

```

let chartSettings: ChartSettings = {
  chartSeries: {
    type: 'Column'
  },
  zoomSettings: {
    enableDeferredZooming: true,
    enableMouseWheelZooming: true,
    enablePinchZooming: true,
    enableSelectionZooming: true
  }
} as ChartSettings;
let dataSourceSettings: IDataOptions = {
  columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
  dataSource: pivotData as IDataset[],
  expandAll: false,
  filters: [],
  formatSettings: [{ name: 'Amount', format: 'C0' }],
  rows: [{ name: 'Country' }, { name: 'Products' }],
  values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
}
let pivotObj: PivotViewComponent;
return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Tooltip

By default, tooltip for the pivot chart is enabled. User can customize it by using the property [tooltipSettings](#) in [chartSettings](#).

The tooltip can be disabled by setting the property [enable](#) in [tooltipSettings](#) as **false**.

INDEX.JSX

```

import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    chartSeries: {
      type: 'Column'
    },
    tooltip: {
      enableMarker: true,
      textStyle: { color: '#000' },
      fill: '#FFF',

```

```

        opacity: 1,
        border: { color: '#000' }
    }
};
let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }, { name: 'Products', type: 'Count' }]
};
let pivotObj;
return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
        chartSeries: {
            type: 'Column'
        },
        tooltip: {
            enableMarker: true,
            textStyle: { color: '#000' },
            fill: '#FFF',
            opacity: 1,
            border: { color: '#000' }
        }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],

```

```

    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }, { name: 'Products', type: 'Count'}]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Export

The pivot chart can be exported using the [chartExport](#) method which holds parameters like export type, file name, PDF orientation, width, and height in the same order. The mandatory parameters for this method are export type and file name whereas other parameters are optional.

The following are the four export types:

- PNG
- JPEG
- SVG
- PDF

In the following code sample, exporting can be done using an external button named as "Chart Export".

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let displayOption = {
    view: 'Chart'
  };
  let chartSettings = {
    chartSeries: { type: 'Column' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' } ]
  };
  let pivotObj;

```

```

function exportClick() {
    pivotObj.chartExport('PNG', 'result');
}
return (<div><div><ButtonComponent cssClass='e-primary'
onClick={exportClick.bind(this)}>Export</ButtonComponent></div><div><PivotVi
ewComponent height={350} ref={d => pivotObj = d} id='PivotView'
chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent></div></div>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
    let displayOption: DisplayOption = {
        view: 'Chart'
    } as DisplayOption;
    let chartSettings: ChartSettings = {
        chartSeries: { type: 'Column' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    function exportClick(): void {
        pivotObj.chartExport('PNG', 'result');
    }
    return (<div><div><ButtonComponent cssClass='e-primary'
onClick={exportClick.bind(this)}>Export</ButtonComponent></div><div><PivotVi
ewComponent height={350} ref={d => pivotObj = d} id='PivotView'
chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}></PivotViewComponent></div></div>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Print

The rendered pivot chart can be printed directly from the browser by calling [printChart](#) method.

In the following code sample, printing can be done using an external button named as "Print".

INDEX.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, Inject, PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let displayOption = {
        view: 'Chart'
    };
    let chartSettings = {
        chartSeries: { type: 'Column' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div><ButtonComponent cssClass='e-primary'
onClick={printClick.bind(this)}>Print</ButtonComponent></div><div><PivotView
Component height={350} ref={d => pivotObj = d} id='PivotView'
chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings}><Inject
services={[PivotChart]}></PivotViewComponent></div></div>);
    function printClick() {
        pivotObj.printChart();
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, Inject, DisplayOption,
PivotChart } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartSettings';
function App() {
```

```

let displayOption: DisplayOption = {
  view: 'Chart'
} as DisplayOption;
let chartSettings: ChartSettings = {
  chartSeries: { type: 'Column' }
} as ChartSettings;
let dataSourceSettings: IDataOptions = {
  columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
  dataSource: pivotData as IDataset[],
  expandAll: false,
  filters: [],
  formatSettings: [{ name: 'Amount', format: 'C0' }],
  rows: [{ name: 'Country' }, { name: 'Products' }],
  values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
}
let pivotObj: PivotViewComponent;
return (<div><div><ButtonComponent cssClass='e-primary'
onClick={printClick.bind(this)}>Print</ButtonComponent></div><div><PivotView
Component height={350} ref={d => pivotObj = d} id='PivotView'
chartSettings={chartSettings} displayOption={displayOption}
dataSourceSettings={dataSourceSettings} ><Inject
services={[PivotChart]}></PivotViewComponent></div></div>);
function printClick(): void {
  pivotObj.printChart();
}
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Drill down in React Pivotview component

Drill down and drill up

The drill down and drill up action helps to view the bound data in detailed and abstract view respectively. By default, if member(s) has children, then expand and collapse icon will be displayed in the respective row/column header. On clicking the icon, expand or collapse action will be performed automatically through built-in source code. Meanwhile, leaf member(s) does not contain expand and collapse icon.

	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
▼ France	450	\$714,955	526	\$1,542,104	
Mountain Bikes	197	\$335,688	238	\$405,552	
Road Bikes	253	\$379,267	288	\$1,136,552	
► Germany	440	\$563,515	496	\$1,772,104	
► United States	546	\$754,515	636	\$2,263,104	
Grand Total	1436	\$2,032,985	1658	\$5,577,312	1

Drill position

Allows to drill only the current position of the selected member and exclude the drilled data of selected member in other positions. For example, if "FY 2015" and "FY 2016" have "Quarter 1" member as child in next level, and when end user attempts to drill "Quarter 1" under "FY 2016", only it will be expanded and not "Quarter 1" under "FY 2015".

This feature is built-in and occurs every time when expand or collapse action is done for better performance.

	FY 2015		FY 2016		
	Q1	FY 2015 Tot...	Q1		
			Mountain Bi...	Road Bikes	Q1 Total
France	114	114	27	67	
Germany	74	74	97	57	
United States	144	144	57	97	
Grand Total	332	332	181	221	

Expand all

This property is applicable only for the relational data source.

Allows to either expand or collapse all headers that are displayed in row and column axes. To display all headers in expanded state, set the property [expandAll](#) to **true** and to collapse all headers, set the property [expandAll](#) to **false**. By default, [expandAll](#) property is set to **false**.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        columns: [{ name: 'Year' }, { name: 'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: true,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    columns: [{name: 'Year' }, {name: 'Quarter'}],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Expand all headers for specific fields

This property is applicable only for the relational data source.

Allows to expand or collapse all headers for specific fields (only) in row and column axes. To expand headers for a specific field in row or column axis, set the property [expandAll](#) in [rows](#) or [columns](#) to **true**. By default, [expandAll](#) property in [rows](#) or [columns](#) is set to **false**.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country', expandAll: true }, { name: 'Products' }],
    columns: [{ name: 'Year', expandAll: true }, { name: 'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
```

```

    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country', expandAll: true }, { name: 'Products' }],
        columns: [{name: 'Year', expandAll: true }, {name: 'Quarter'}],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Expand all except specific member(s)

This option is applicable only for the relational data source.

In addition to the previous topic, there is an enhancement to expand all headers except specific header(s) and similarly to collapse all headers except specific header(s). To achieve this, [drilledMembers](#) is used. The required properties of the [drilledMembers](#) are explained below:

- [name](#): It allows to set the field name whose member(s) needs to be specifically drilled.
- [items](#): It allows to set the exact member(s) which needs to be drilled.

The [drilledMembers](#) option always works in vice-versa with respect to the property [expandAll](#) in pivot table. For example, if [expandAll](#) is set to **true**, then the member(s) added in [items](#) collection alone will be in collapsed state.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';

```

```
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: true,
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataSet, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataSet[],
    expandAll: true,
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Expand specific member(s)

End user can also manually expand or collapse specific member(s) in each fields under row and column axes using the [drilledMembers](#) from code behind. The required properties of the [drilledMembers](#) are explained below:

- [name](#): It allows to set the field name whose member(s) needs to be specifically drilled.
- [items](#): It allows to set the exact member(s) which needs to be drilled.
- [delimiter](#): It allows to separate next level of member from its parent member.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
```

```
ReactDOM.render(<App />, document.getElementById('sample'));
```

Event

Drill

The event [drill](#) triggers every time when a field is expanded or collapsed. For instance using this event user can alter delimiter and drill action for the respective item. It has the following parameters:

- **drillInfo** - It holds the current drilled item information.
- **pivotview** - It holds pivot table instance.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C2' }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    };
    let pivotObj;
    function drill(args) {
        args.dataSourceSettings.columns[0].caption = "Full Year";
        args.dataSourceSettings.expandAll = true;
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} drill={drill.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent, DrillArgs } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
```

```

    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France'] }],
    formatSettings: [{ name: 'Amount', format: 'C2'}],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
  }
  let pivotObj: PivotViewComponent;
  function drill(args: DrillArgs): void {
    args.dataSourceSettings.columns[0].caption = "Full Year";
    args.dataSourceSettings.expandAll = true;
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} drill={drill.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

ActionBegin

The event [actionBegin](#) triggers when the UI actions such as drill down and drill up begin. This allows user to identify the current action being performed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action began. The following are the UI actions and their names:

| Action | Action Name |

|-----|-----|

| [Expand](#) | Drill down |

| [Collapse](#) | Drill up |

- **cancel**: It allows user to restrict the current action.

In the below sample, drill down and drill up action can be restricted by setting the **args.cancel** option to **true** in the **actionBegin** event.

INDEX.JSX

```

import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';

```

```
function App() {
  let dataSourceSettings = {
    dataSource: pivotData,
    expandAll: false,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    filters: [],
  };
  let pivotObj;
  function actionBegin(args) {
    if (args.actionName == 'Drill down' || args.actionName == 'Drill
up') {
      args.cancel = true;
    }
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
actionBegin={actionBegin.bind(this)} showFieldList={true}><Inject
services={[FieldList]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent,
PivotActionBeginEventArgs } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: false,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    filters: [],
  }
  let pivotObj: PivotViewComponent;
  function actionBegin(args: PivotActionBeginEventArgs): void {
    if (args.actionName == 'Drill down' || args.actionName == 'Drill
up') {
      args.cancel = true;
    }
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
actionBegin={actionBegin.bind(this)} showFieldList={true} ><Inject
services={[FieldList]} /></PivotViewComponent>);
```



```
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

ActionComplete

The event [actionComplete](#) triggers when a UI action such as drill down or drill up, is completed. This allows user to identify the current UI actions being completed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action completed. The following are the UI actions and their names:

Action	Action Name
-----	-----
Expand	Drill down
Collapse	Drill up

- **actionInfo**: It holds the unique information about the current UI action. For example, if drill down action is completed, the event argument contains information such as field name and the drill information.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        dataSource: pivotData,
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }],
        filters: [],
    };
    let pivotObj;
    function actionComplete(args) {
        if (args.actionName == 'Drill down' || args.actionName == 'Drill up') {
            // Triggers when the drill operations are completed.
        }
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings}
```

```

actionComplete={actionComplete.bind(this)} showFieldList={true}><Inject
services={[FieldList]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent,
PivotActionCompleteEventArgs } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: false,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    filters: [],
  }
  let pivotObj: PivotViewComponent;
  function actionComplete(args: PivotActionCompleteEventArgs): void {
    if (args.actionName == 'Drill down' || args.actionName == 'Drill up')
    {
      // Triggers when the drill operations are completed.
    }
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
actionComplete={actionComplete.bind(this)} showFieldList={true} ><Inject
services={[FieldList]} /></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

ActionFailure

The event [actionFailure](#) triggers when the current UI action fails to achieve the desired result. It has the following parameters:

- actionName:** It holds the name of the current action failed. The following are the UI actions and their names:

Action	Action Name
-----	-----
Expand	Drill down
Collapse	Drill up

- **errorInfo**: It holds the error information of the current UI action.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        dataSource: pivotData,
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }],
        filters: [],
    };
    let pivotObj;
    function actionFailure(args) {
        if (args.actionName == 'Drill down' || args.actionName == 'Drill up') {
            // Triggers when the current UI action fails to achieve the desired result.
        }
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} actionFailure={actionFailure.bind(this)} showFieldList={true}><Inject services={[FieldList]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent, PivotActionFailureEventArgs } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }],
        filters: [],
    }
    let pivotObj: PivotViewComponent;
```

```
function actionFailure(args: PivotActionFailureEventArgs): void {
    if (args.actionName == 'Drill down' || args.actionName == 'Drill up')
    {
        // Triggers when the current UI action fails to achieve the
        desired result.
    }
}

return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
actionFailure={actionFailure.bind(this)} showFieldList={true} ><Inject
services={[FieldList]} /></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Data Shaping

Aggregation in React Pivotview component

This feature is applicable only for the relational data source.

End user can perform calculations over a group of values (exclusively for value fields bound in value axis) using the aggregation option. By default, values are added (summed) together. The other aggregation types are explained below.

The fields with data type such as number support all aggregation types mentioned below except for **“CalculatedField”**. The fields with data type such as string, date, datetime, boolean, etc., support **“Count”** and **“DistinctCount”** aggregation types alone.

Operator	Description
----- -----	
Sum	Displays the pivot table values with sum.
Product	Displays the pivot table values with product.
Count	Displays the pivot table values with count.
DistinctCount	Displays the pivot table values with distinct count.
Min	Displays the pivot table with minimum value.
Max	Displays the pivot table with maximum value.
Avg	Displays the pivot table values with average.
Median	Displays the pivot table values with median.
Index	Displays the pivot table values with index.
PopulationStDev	Displays the pivot table values with standard deviation of population.
SampleStDev	Displays the pivot table values with sample standard deviation.
PopulationVar	Displays the pivot table values with variance of population.
SampleVar	Displays the pivot table values with sample variance.
RunningTotals	Displays the pivot table values with running totals.

| **DifferenceFrom** | Displays the pivot table values with difference from the value of the base item in the base field. |

| **PercentageOfDifferenceFrom** | Displays the pivot table values with percentage difference from the value of the base item in the base field. |

| **PercentageOfGrandTotal** | Displays the pivot table values with percentage of grand total of all values. |

| **PercentageOfColumnTotal** | Displays the pivot table values in each column with percentage of total values for the column. |

| **PercentageOfRowTotal** | Displays the pivot table values in each row with percentage of total values for the row. |

| **PercentageOfParentTotal** | Displays the pivot table values with percentage of total of all values based on selected field. |

| **PercentageOfParentColumnTotal** | Displays the pivot table values with percentage of its parent total in each column. |

| **PercentageOfParentRowTotal** | Displays the pivot table values with percentage of its parent total in each row. |

| **CalculatedField** | Displays the pivot table with calculated field values. It allows user to create a new calculated field alone. |

Assigning aggregation type for value fields through API

For each value field, the aggregation type can be set using the property [type](#) in [values](#) class. Meanwhile, aggregation types like **DifferenceFrom** and **PercentageOfDifferenceFrom** can check for specific field of specific item using [baseField](#) and [baseItem](#) properties. Likewise, **PercentageOfParentTotal** type can for specific field using [baseField](#) property. For instance, the aggregation type **DifferenceFrom** would intake the specified field and its corresponding member as input and its value is compared across other members in the same field and also across different fields to formulate an appropriate output value.

- [type](#): It allows to set the aggregate type of the field.
- [baseField](#): It allows to set the specific field to aggregate the values.
- [baseItem](#): It allows to set the specific member to aggregate the values.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount', type: 'Sum' }]
```

```

    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
  }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount', type: 'Sum' } ]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

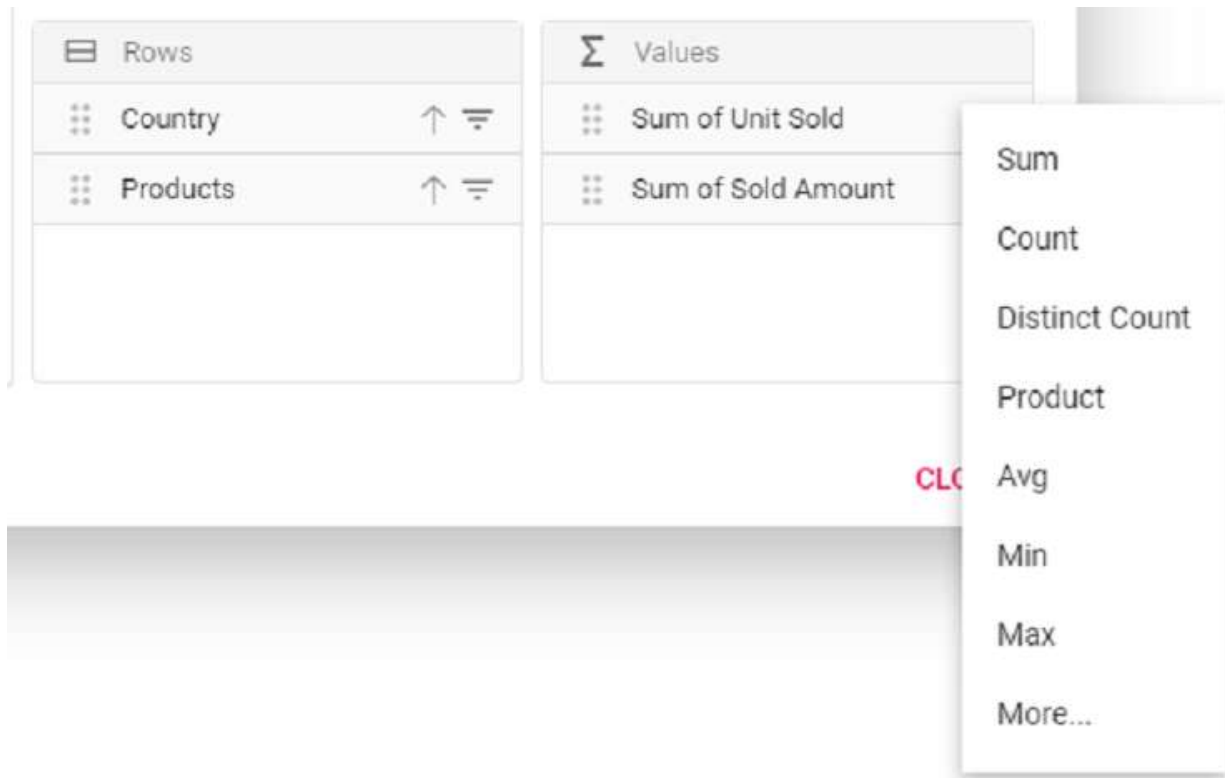
```

By default, the aggregation will be considered as **Sum** to the value fields which had number type and for the value fields which had non-number type values such as string, date, datetime, boolean, etc., the aggregation type will be considered as **Count**.

Modifying aggregation type for value fields at runtime

Aggregation types can be changed easily through UI at runtime. The value fields bound to grouping bar and field list appears with a dropdown icon which helps to select an appropriate aggregation type for the respective value field. On selection, the values in the pivot table will be changed dynamically.

<!-- markdownlint-disable MD012 -->



Sum of Units Sold	Sum	Drop filter here		
Sum of Sold Amount	Count	Year	Quarter	
Country	Distinct Count	FY 2015		FY 2016
Products	Product	Units Sold	Sold Amount	Units Sold
▶ France	Min	450	\$714,955	526
▶ Germany	Max	440	\$563,515	496
▶ United States	Avg	546	\$754,515	636
Grand Total	More...	1436	\$2,032,985	1658

Show desired aggregation types in its dropdown menu

By default, all the aggregation types are displayed in the dropdown menu available in buttons. However, based on the request for an application, we may need to show selective aggregation types on our own. This can be achieved using the [aggregateTypes](#) property.

INDEX.JSX

```

import { GroupingBar, FieldList, Inject, PivotViewComponent } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        dataSource: pivotData,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount', type: 'Sum' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        filters: [],
    };
    let pivotObj;
    let pivotAggregateTypes = ['DistinctCount', 'Avg', 'Product'];
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}
showGroupingBar={true} aggregateTypes={pivotAggregateTypes}><Inject
services={[GroupingBar, FieldList]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { GroupingBar, FieldList, Inject, IDataOptions, IDataset,
PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataset,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount', type: 'Sum' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        filters: [],
    }
    let pivotObj: PivotViewComponent;
    let pivotAggregateTypes: any = ['DistinctCount', 'Avg', 'Product'];
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}
showGroupingBar={true} aggregateTypes={pivotAggregateTypes}><Inject
services={[GroupingBar, FieldList]}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```


Hiding aggregation type from button text

By default, in value axis each field would be displayed by its name and aggregation type together. To hide aggregation type and display field name alone, set the property [showAggregationOnValueField](#) in [dataSourceSettings](#) to **false**.

INDEX.JSX

```
import { PivotViewComponent, Inject, GroupingBar } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount', type: 'Sum' }],
        showAggregationOnValueField: false
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} showGroupingBar={true}><Inject services={[GroupingBar]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent, Inject, GroupingBar } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount', type: 'Sum' }],
        showAggregationOnValueField: false
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} showGroupingBar={true}><Inject services={[GroupingBar]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

```

    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showGroupingBar={true}><Inject
services={[GroupingBar]}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Hiding aggregation type icon from UI

By default, the icon to set aggregation type is enabled in the grouping bar. To disable this icon, set the property [showValueTypeIcon](#) in [groupingBarSettings](#) to **false**.

Icon to change the aggregation type can be hidden only in Grouping Bar but not in Field List at the moment.

INDEX.JSX

```

import { GroupingBar, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let groupingSettings = {
        showValueTypeIcon: false
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} groupingBarSettings={groupingSettings}
dataSourceSettings={dataSourceSettings} showGroupingBar={true}><Inject
services={[GroupingBar]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { GroupingBar, GroupingBarSettings, IDataOptions, IDataset, Inject,
PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';

```

```
function App() {
  let groupingSettings: GroupingBarSettings = {
    showValueTypeIcon: false
  } as GroupingBarSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} groupingBarSettings={groupingSettings}
dataSourceSettings={dataSourceSettings} showGroupingBar={true} ><Inject
services={[GroupingBar]}/> </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Event

AggregateCellInfo

The event [aggregateCellInfo](#) triggers every time while rendering each value cell. This allows user to change the cell value and skip formatting if applied. It has following parameters:

- **fieldName** - It holds current cell's field name.
- **row** - It holds current cell's row value.
- **column** - It holds current cell's row value.
- **value** - It holds value of current cell.
- **cellSets** - It holds raw data for the aggregated value cell.
- **rowCellType** - It holds row cell type value.
- **columnCellType** - It holds column cell type value.
- **aggregateType** - It holds aggregate type of the cell.
- **skipFormatting** - boolean property, it allows to skip formatting if applied.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
```

```

        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C2' }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    };
    let pivotObj;
    function aggregateCell(args) {
        args.skipFormatting = true;
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} aggregateCellInfo={aggregateCell.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, AggregateEventArgs }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C2' }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    }
    let pivotObj: PivotViewComponent;
    function aggregateCell(args: AggregateEventArgs): void {
        args.skipFormatting = true;
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} aggregateCellInfo={aggregateCell.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

ActionBegin

The event [actionBegin](#) triggers when clicking and selecting the aggregate type via the dropdown icon in the value field button, which is present in both grouping bar and field list UI. This allows user to identify the current action being performed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action began. For example, while performing aggregation, the action name will be shown as **Aggregate field**.
- **fieldInfo**: It holds the selected value field information.

Note: This option is applicable only when the field based UI actions are performed such as filtering, sorting, removing field from grouping bar, editing and aggregation type change.

- **cancel**: It allows user to restrict the current action.

In the following example, action taken during aggregation type selection via dropdown icon can be restricted by setting the **args.cancel** option to **true** in the **actionBegin** event.

INDEX.JSX

```
import { FieldList, GroupingBar, Inject, PivotViewComponent } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    dataSource: pivotData,
    expandAll: false,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
 'Quarter' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
 caption: 'Sold Amount' }],
    filters: [],
  };
  let pivotObj;
  function actionBegin(args) {
    if (args.actionName == 'Aggregate field') {
      args.cancel = true;
    }
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
 height={350} dataSourceSettings={dataSourceSettings}
 actionBegin={actionBegin.bind(this)} showFieldList={true}
 showGroupingBar={true}><Inject services={[FieldList,
 GroupingBar]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { FieldList, GroupingBar, IDataOptions, IDataset, Inject,
PivotViewComponent, PivotActionBeginEventArgs } from '@syncfusion/ej2-react-
pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: false,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    filters: [],
  }
  let pivotObj: PivotViewComponent;
  function actionBegin(args: PivotActionBeginEventArgs): void {
    if (args.actionName == 'Aggregate field') {
      args.cancel = true;
    }
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
actionBegin={actionBegin.bind(this)} showFieldList={true}
showGroupingBar={true}><Inject services={[FieldList, GroupingBar]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

ActionComplete

The event [actionComplete](#) triggers when a UI action, such as applying aggregation using the dropdown icon via the value field button, which is present in both the grouping bar and the field list UI, is completed. This allows user to identify the current UI action being completed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action completed. For example, after completing the aggregation, the action name will be shown as **Field aggregated**.
- **fieldInfo**: It holds the selected value field information.

Note: This option is applicable only when the field based UI actions are performed such as filtering, sorting, removing field from grouping bar, editing and aggregation type change.

INDEX.JSX

```
import { FieldList, GroupingBar, Inject, PivotViewComponent } from
'@syncfusion/ej2-react-pivotview';
import * as React from 'react';
```

```

import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        dataSource: pivotData,
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        filters: [],
    };
    let pivotObj;
    function actionComplete(args) {
        if (args.actionName == 'Field aggregated') {
            // Triggers when the aggregation type is applied.
        }
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
actionComplete={actionComplete.bind(this)} showFieldList={true}
showGroupingBar={true}><Inject services={[FieldList,
GroupingBar]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { FieldList, GroupingBar, IDataOptions, IDataset, Inject,
PivotViewComponent, PivotActionCompleteEventArgs } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        filters: [],
    }
    let pivotObj: PivotViewComponent;
    function actionComplete(args: PivotActionCompleteEventArgs): void {
        if (args.actionName == 'Field aggregated') {
            // Triggers when the aggregation type is applied.
        }
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}

```

```

actionComplete={actionComplete.bind(this)} showFieldList={true}
showGroupingBar={true}><Inject services={[FieldList, GroupingBar]}
/></PivotViewComponent>);

};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

ActionFailure

The event [actionFailure](#) triggers when the current UI action fails to achieve the desired result. It has the following parameters:

- **actionName**: It holds the name of the current action failed. For example, if the action fails while performing the aggregation, then the action name will be shown as **Aggregate field**.
- **errorInfo**: It holds the error information of the current UI action.

INDEX.JSX

```

import { FieldList, GroupingBar, Inject, PivotViewComponent } from
'@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        dataSource: pivotData,
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        filters: [],
    };
    let pivotObj;
    function actionFailure(args) {
        if (args.actionName == 'Aggregate field') {
            // Triggers when the current UI action fails to achieve the
desired result.
        }
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
actionFailure={actionFailure.bind(this)} showFieldList={true}
showGroupingBar={true}><Inject services={[FieldList,
GroupingBar]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX


```
import { FieldList, GroupingBar, IDataOptions, IDataset, Inject,
PivotViewComponent, PivotActionFailureEventArgs } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: false,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    filters: [],
  }
  let pivotObj: PivotViewComponent;

  function actionFailure(args: PivotActionFailureEventArgs): void {
    if (args.actionName === 'Aggregate field') {
      // Triggers when the current UI action fails to achieve the
desired result.
    }
  }

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
actionFailure={actionFailure.bind(this)} showFieldList={true}
showGroupingBar={true}><Inject services={[FieldList, GroupingBar]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Calculated field in React Pivotview component

Allows end user to create a new calculated field in the pivot table, based on available fields from the bound data source or using simple formula with basic arithmetic operators. It can be added at runtime through the built-in dialog, invoked from Field List UI. To do so, set the [allowCalculatedField](#) property to **true** in the pivot table. End user can now see a "CALCULATED FIELD" button enabled in Field List UI automatically, which on clicking will invoke the calculated field dialog and perform necessary operation.

Calculated field can also be included in the pivot table through code behind using the [calculatedFieldsSettings](#). The required properties to create a new calculate field are:

- [name](#): It allows to indicate the calculated field with a unique name.
- [formula](#): It allows to set the formula.
- [formatSettings](#): It helps to set the number format for the resultant value.

To use calculated field option, you need to inject the `CalculatedField` module in pivot table.

The calculated field is applicable only for value fields. Also, calculated field created through code behind will be automatically listed in the UI dialog as well.

INDEX.JSX

```

import { CalculatedField, FieldList, Inject, PivotViewComponent } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }, { name: 'Total',
format: 'C2' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }, { name: 'Total', caption: 'Total Units', type:
 'CalculatedField' }],
        calculatedFieldSettings: [{ name: 'Total', formula:
 '"Sum(Amount)"+"Sum(Sold)"' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}><Inject
services={[CalculatedField, FieldList]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { CalculatedField, FieldList, IDataOptions, IDataset, Inject,
PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
 'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }, { name: 'Total',
format: 'C2' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }, { name: 'Total', caption: 'Total Units', type:
 'CalculatedField' }],
        calculatedFieldSettings: [{ name: 'Total', formula:
 '"Sum(Amount)"+"Sum(Sold)"' }]
    }
}

```

```

    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}><Inject
services=[CalculatedField, FieldList]/> </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Meanwhile, user can also view calculated field dialog in UI by invoking [createCalculatedFieldDialog](#) method on an external button click which is shown in the below code sample.

To use calculated field option, you need to inject the **CalculatedField** module in pivot table.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { CalculatedField, Inject, PivotViewComponent } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }, { name: 'Total',
format: 'C2' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }, { name: 'Total', caption: 'Total Units', type:
'CalculatedField' }],
        calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount)"+"Sum(Sold)"' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings} allowCalculatedField={true}><Inject
services=[CalculatedField]/> </PivotViewComponent></div>
    <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary' onClick={btnClick.bind(this)}>Calculated
Field</ButtonComponent></div></div>);
    function btnClick() {
        pivotObj.calculatedFieldModule.createCalculatedFieldDialog();
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

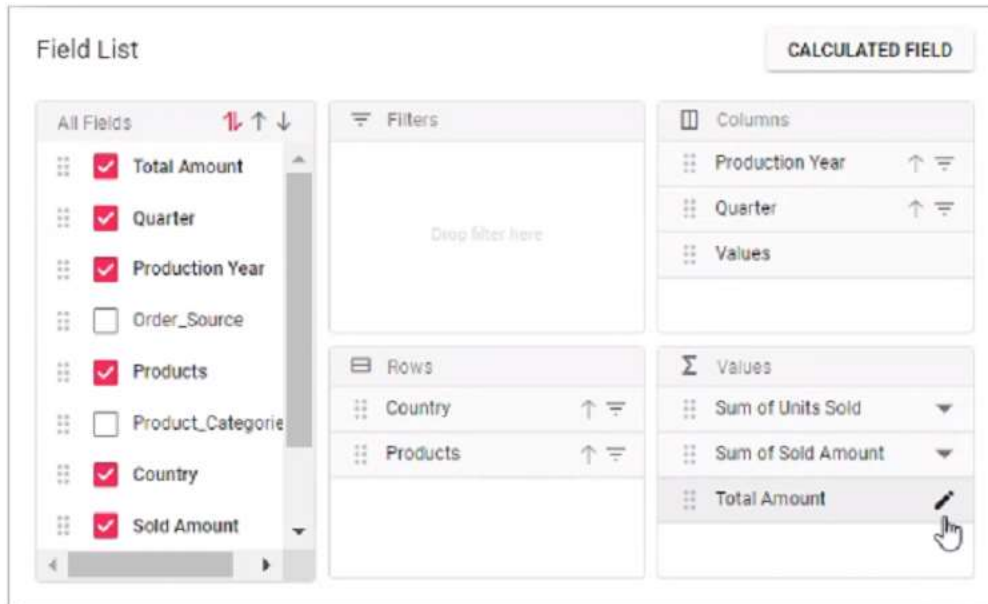
```

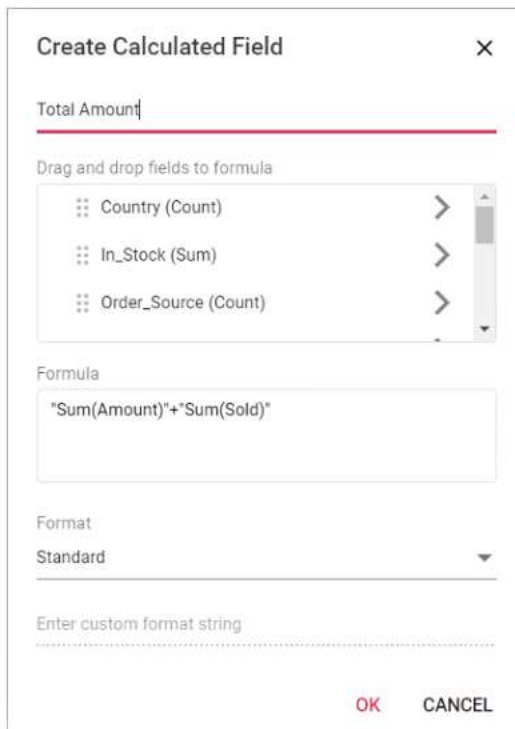
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { CalculatedField, IDataOptions, IDataset, Inject, PivotViewComponent
} from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }, { name: 'Total',
format: 'C2' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }, { name: 'Total', caption: 'Total Units', type:
'CalculatedField' }],
    calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount)"+"Sum(Sold) "' }]]
  }
  let pivotObj: PivotViewComponent;
  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings} allowCalculatedField={true}><Inject
services={[CalculatedField]}/> </PivotViewComponent></div>
  <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary' onClick={btnClick.bind(this)}>Calculated
Field</ButtonComponent></div></div>);
  function btnClick(): void {
    pivotObj.calculatedFieldModule.createCalculatedFieldDialog();
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Editing through the field list and the grouping bar

User can also modify the existing calculated field using the built-in edit option available directly in the field list (or) grouping bar. To do so, click the "Edit" icon available in the calculated field button. Now the calculated field dialog is opened and the current calculated field name, formula and format can be changed at runtime.





Renaming the existing calculated field

Existing calculated field can be renamed only through the UI at runtime. To do so, open the calculated field dialog, select the target field and click "Edit" icon. User can now see the existing name getting displayed in the text box at the top of the dialog. Now, change the name based on user requirement and click "OK".

<!-- markdownlint-disable MD012 -->

Create Calculated Field ×

Enter the field name

Drag and drop fields to formula

- Sold Amount (Sum)
- Total Amount (Calculated Field)** Edit
- Units Sold (Sum)

Formula

Example: ("Sum(Order_Count)" + "Sum(In_Stock)") * 250

Format

None

Enter custom format string

OK **CANCEL**

Create Calculated Field ×

Total Sales Amount

Drag and drop fields to formula

- Sold Amount (Sum)
- Total Amount (Calculated Field)** Edit
- Units Sold (Sum)

Formula

"Sum(Amount)"+"Sum(Sold)"

Format

Standard

Enter custom format string

OK **CANCEL**

Editing the existing calculated field formula

Existing calculated field formula can be edited only through the UI at runtime. To do so, open the calculated field dialog, select the target field and click "Edit" icon. User can now see the existing formula

getting displayed in a multiline text box at the bottom of the dialog. Now, change the formula based on user requirement and click "OK".

Create Calculated Field [X]

Enter the field name

Drag and drop fields to formula

- Sold Amount (Sum) >
- Total Amount (Calculated Field) [trash] [edit]
- Units Sold (Sum) >

Formula

Example: ("Sum(Order_Count)" + "Sum(In_Stock)") * 250

Format

None

Enter custom format string

OK CANCEL

Create Calculated Field [X]

Total Amount

Drag and drop fields to formula

- Sold Amount (Sum) >
- Total Amount (Calculated Field) [trash] [edit]
- Units Sold (Sum) >

Formula

"Sum(Amount)"+"Sum(Sold)"

Format

Standard

Enter custom format string

OK CANCEL

Reusing the existing formula in a new calculate field

While creating a new calculated field, if user wants to add the formula of an existing calculated field, it can be done easily. To do so, simply drag-and-drop the existing calculated field to the "Formula" section.

The screenshot shows the 'Create Calculated Field' dialog box. At the top, there's a title bar with 'Create Calculated Field' and a close button. Below it, the 'Sales Amount' field is visible. The 'Drag and drop fields to formula' section contains a list of fields: 'Sold Amount (Sum)', 'Total Amount (Calculated Field)', and 'Total Amount (Calculated Field)'. The 'Total Amount (Calculated Field)' is highlighted. Below this list is the 'Formula' section, which contains a text box with the example formula: 'Example: ("Sum(Order_Count)" + "Sum(In_Stock)") * 250'. Below the formula section is the 'Format' section, which has a dropdown menu set to 'None'. At the bottom, there is a field for 'Enter custom format string' and two buttons: 'OK' and 'CANCEL'.

Create Calculated Field

Sales Amount

Drag and drop fields to formula

Sold Amount (Sum)

Total Amount (Calculated Field)

Units Sold (Sum)

Formula

Example: ("Sum(Order_Count)" + "Sum(In_Stock)") * 250

Total Amount (Calculated Field)

Format

None

Enter custom format string

OK

CANCEL

Create Calculated Field

Sales Amount

Drag and drop fields to formula

Sold Amount (Sum)

Total Amount (Calculated Field)

Units Sold (Sum)

Formula

"Sum(Amount)"+"Sum(Sold)"

Format

None

Enter custom format string

OK

CANCEL

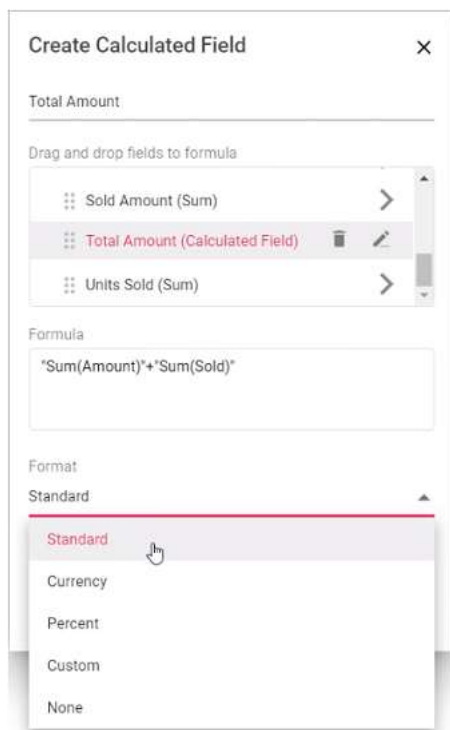
Apply the format to the calculated field values

Values in a new or existing calculated field can be formatted via the calculated field UI or code behind. The [formatSettings](#) property in code-behind can be used to specify the desired format. For more information about the supported formats refer [here](#).

To apply format to calculated field values at runtime via UI, a built-in dropdown under the "Format" label is available, from which the user can select the pre-defined format options listed below.

- **Standard** - Denotes the numeric type.
- **Currency** - Denotes the currency type.
- **Percent** - Denotes the percentage type.
- **Custom** - Denotes the custom format. For example: "C2". This shows the value "9584.3" as "\$9584.30."
- **None** - Denotes that no format will be applied.

By default, **None** will be selected from the dropdown.



In addition, you can specify the desired custom formats by selecting the **Custom** option from the "Format" dropdown.

Create Calculated Field [X]

Sales Amount

Drag and drop fields to formula

- Sold Amount (Sum) >
- Total Amount (Calculated Field) [trash] [edit]
- Units Sold (Sum) >

Formula

"Sum(Amount)"+"Sum(Sold)"

Format

Custom

C2 I

OK CANCEL

Supported operators and functions for the calculated field formula

Below is a list of operators and functions that can be used in the formula to create the calculated fields.

- **+** – addition operator.

`ts

Syntax: X + Y

,

- **-** – subtraction operator.

`ts

Syntax: X - Y

,

- ***** – multiplication operator.

`ts

Syntax: X * Y

,

- **/** – division operator.

`ts

Syntax: X / Y

,

- `^` – power operator.

Grouping in React Pivotview component

This feature is applicable only for relational data source.

Grouping is the most-useful feature in pivot table and the component automatically groups date, time, number and string. For example, the date type can be formatted and displayed based on year, quarter, month, and more. Likewise, the number type can be grouped range-wise, such as 1-5, 6-10, etc. These group fields will act as individual fields and allows users to drag them between different axes such as columns, rows, values, and filters and create pivot table at runtime.

<!-- markdownlint-disable MD012 -->

Filtering in React Pivotview component

Filtering allows to view the pivot table with selective records based on members that can be either included or excluded through UI and code-behind.

The following are the three different types of filtering:

- Member filtering
- Label filtering
- Value filtering

When all the above filtering options are disabled via code-behind, then the filter icon would be disabled in the field list or grouping bar UI.

Member filtering

Sorting in React Pivotview component

To have a quick glance on how to sort data in the React Pivot Table, watch this video:

Drill through in React Pivotview component

Editing in React Pivotview component

This feature is applicable only for the relational data source.

Data Formatting

Number formatting in React Pivotview component

Allows you to specify the required display format that should be used in values of the pivot table.

Supported display formats are:

- Number
- Currency
- Percentage
- Custom

Conditional formatting in React Pivotview component

Allows end user to change the appearance of the pivot table value cells with its background color, font color, font family, and font size based on specific conditions.

Report Manipulation

Grouping bar in React Pivotview component

To have a quick glance on how to enable grouping bar in the React Pivot Table, watch this video:

Field list in React Pivotview component

To have a quick glance on how to enable field list in the React Pivot Table, watch this video:

Defer update in React Pivotview component

Defer layout update support allows to update the pivot table component only on demand. On enabling this feature, end user can drag-and-drop fields between row, column, value and filter axes, apply sorting and filtering inside the Field List, resulting in change of pivot report alone but not the pivot table values. Once all operations are performed and on clicking the "Apply" button in the Field List, pivot table will start to update with the last modified report. This also helps in bringing better performance in pivot table component rendering.

The field list can be displayed in two different formats to interact with pivot table. They are:

- **In-built Field List (Popup):** To display the field list icon in pivot table UI to invoke the built-in dialog.
- **Stand-alone Field List (Fixed):** To display the field list in a static position within a web page.

In-built Field List (Popup)

To enable deferred updates in the pivot table, set the property [allowDeferLayoutUpdate](#) as **true** in [PivotView](#). To make a note, the defer update option can be controlled only via Field List during runtime.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    allowLabelFilter: true,
    allowValueFilter: true,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}
```

```
allowDeferLayoutUpdate={true}><Inject
services={[FieldList]}/></PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    allowLabelFilter: true,
    allowValueFilter: true,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}
allowDeferLayoutUpdate={true}><Inject services={[FieldList]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Stand-alone Field List (Fixed)

The field list can be rendered in a static position, anywhere in web page layout, like a separate component. To do so, you need to set [renderMode](#) property to **Fixed** in PivotFieldList.

To enable deferred updates in the static fieldlist, set the property [allowDeferLayoutUpdate](#) in [PivotFieldList](#) as **true**. To make a note, the defer update option can be controlled only via Field List during runtime.

To make field list interact with pivot table, you need to use the **UpdateView** and **Update** methods for data source update in both field list and pivot table simultaneously.

INDEX.JSX

```
{% raw %}
import { CalculatedField, PivotFieldListComponent, Inject,
PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
```

```

import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
const SAMPLE_CSS = `
.e-pivotview {
  width: 58%;
  height: 100%;
  float: left;
}
.e-pivotfieldlist {
  width: 42%;
  height: 100%;
  float: right;
}
.e-pivotfieldlist .e-static {
  width: 100% !important;
} `;
function App() {
  React.useEffect(() => {
    renderComplete();
  }, []);
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  let fieldlistObj;
  return (<div className="control-section">
    <style>{SAMPLE_CSS}</style>
    <PivotViewComponent id='PivotViewFieldList' ref={d => pivotObj = d}
enginePopulated={afterPivotPopulate.bind(this)}
allowDeferLayoutUpdate={true} width={'99%'} height={'530'} gridSettings={{
columnWidth: 140 }}></PivotViewComponent>
    <PivotFieldListComponent id='PivotFieldList' ref={d => fieldlistObj = d}
enginePopulated={afterPopulate.bind(this)}
dataSourceSettings={dataSourceSettings} renderMode={"Fixed"}
allowDeferLayoutUpdate={true} allowCalculatedField={true}><Inject
services={[CalculatedField]}></PivotFieldListComponent></div>);
  function afterPopulate() {
    if (fieldlistObj && pivotObj) {
      if (fieldlistObj.isRequiredUpdate) {
        fieldlistObj.updateView(pivotObj);
      }
      pivotObj.notify('ui-update', pivotObj);
      fieldlistObj.notify('tree-view-update', fieldlistObj);
    }
  }
  function afterPivotPopulate() {
    if (fieldlistObj && pivotObj) {
      fieldlistObj.update(pivotObj);
    }
  }
}

```

```

    }
    function renderComplete() {
        fieldlistObj.updateView(pivotObj);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { CalculatedField, PivotFieldListComponent, IDataOptions, IDataset,
Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
const SAMPLE_CSS = `
.e-pivotview {
    width: 58%;
    height: 100%;
    float: left;
}
.e-pivotfieldlist {
    width: 42%;
    height: 100%;
    float: right;
}
.e-pivotfieldlist .e-static {
    width: 100% !important;
}
`;
function App() {
    React.useEffect(() => {
        renderComplete();
    }, []);
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    let fieldlistObj: PivotFieldListComponent;
    return (<div className="control-section">
        <style>{SAMPLE_CSS}</style>
        <PivotViewComponent id='PivotViewFieldList' ref={d => pivotObj = d}
enginePopulated={afterPivotPopulate.bind(this)}
allowDeferLayoutUpdate={true} width={'99%'} height={'530'}
gridSettings={{columnWidth: 140}}></PivotViewComponent>

```



```

    <PivotFieldListComponent id='PivotFieldList' ref={d => fieldlistObj = d}
    enginePopulated={afterPopulate.bind(this)}
    dataSourceSettings={dataSourceSettings} renderMode={"Fixed"}
    allowDeferLayoutUpdate={true} allowCalculatedField={true}><Inject
    services={[CalculatedField]} /></PivotFieldListComponent></div>;
    function afterPopulate(): void {
        if (fieldlistObj && pivotObj) {
            if (fieldlistObj.isRequiredUpdate) {
                fieldlistObj.updateView(pivotObj);
            }
            pivotObj.notify('ui-update', pivotObj);
            fieldlistObj.notify('tree-view-update', fieldlistObj);
        }
    }
    function afterPivotPopulate(): void {
        if (fieldlistObj && pivotObj) {
            fieldlistObj.update(pivotObj);
        }
    }
    function renderComplete(): void {
        fieldlistObj.updateView(pivotObj);
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

Performance

<!-- markdownlint-disable MD036 -->

Virtual scrolling in React Pivot Table component

Virtual Scrolling

Allows to load the large amounts of data without any performance degradation by rendering rows and columns only in the current content viewport. Rest of the aggregated data will be brought into viewport dynamically based on vertical or horizontal scroll position. This feature can be enabled by setting the [enableVirtualization](#) property to **true**.

To use the virtual scrolling feature, inject the **VirtualScroll** module in to the pivot table.

INDEX.JSX

```

import { PivotViewComponent, VirtualScroll, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
let datel1;
let date2;
function data(count) {
    let result = [];
    let dt = 0;
    for (let i = 1; i < (count + 1); i++) {
        dt++;
        let round;
        let toString = i.toString();
        if (toString.length === 1) {

```

```

        round = '0000' + (i);
    }
    else if (toString.length === 2) {
        round = '000' + i;
    }
    else if (toString.length === 3) {
        round = '00' + i;
    }
    else if (toString.length === 4) {
        round = '0' + i;
    }
    else {
        round = toString;
    }
    result.push({
        ProductID: 'PRO-' + round,
        Year: "FY " + (dt + 2013),
        Price: Math.round(Math.random() * 5000) + 5000,
        Sold: Math.round(Math.random() * 80) + 10,
    });
    if (dt / 4 == 1) {
        dt = 0;
    }
}
return result;
}
;
function App() {
    let dataSourceSettings = {
        dataSource: data(1000),
        enableSorting: false,
        expandAll: true,
        formatSettings: [{ name: 'Price', format: 'C0' }],
        rows: [{ name: 'ProductID' }],
        columns: [{ name: 'Year' }],
        values: [{ name: 'Price', caption: 'Unit Price' }, { name: 'Sold',
caption: 'Unit Sold' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enableVirtualization={true}
dataSourceSettings={dataSourceSettings}><Inject
services={[VirtualScroll]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, VirtualScroll, Inject }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
let date1: number;
let date2: number;

```

```

function data(count: number) {
  let result: Object[] = [];
  let dt: number = 0;
  for (let i: number = 1; i < (count + 1); i++) {
    dt++;
    let round: string;
    let toString: string = i.toString();
    if (toString.length === 1) {
      round = '0000' + (i);
    }
    else if (toString.length === 2) {
      round = '000' + i;
    }
    else if (toString.length === 3) {
      round = '00' + i;
    } else if (toString.length === 4) {
      round = '0' + i;
    } else {
      round = toString;
    }
    result.push({
      ProductID: 'PRO-' + round,
      Year: "FY " + (dt + 2013),
      Price: Math.round(Math.random() * 5000) + 5000,
      Sold: Math.round(Math.random() * 80) + 10,
    });
    if (dt / 4 == 1) {
      dt = 0;
    }
  }
  return result;
};

function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: data(1000),
    enableSorting: false,
    expandAll: true,
    formatSettings: [{ name: 'Price', format: 'C0' }],
    rows: [{ name: 'ProductID' }],
    columns: [{ name: 'Year' }],
    values: [{ name: 'Price', caption: 'Unit Price' }, { name: 'Sold',
caption: 'Unit Sold' }]
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enableVirtualization={true}
dataSourceSettings={dataSourceSettings}><Inject
services={ [VirtualScroll] }/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

The **height** and **width** properties should be set for virtual scrolling. If it is not defined then the pivot table will consider its value as **300px** and **800px** respectively.

Virtual scrolling with single page mode

When virtual scrolling is enabled, the pivot table renders not only the current view page, but also the previous and next pages by default. This default behavior, however, can cause performance delays when dealing with a large number of rows and columns. This is because the same number of rows and columns from adjacent pages are also processed, resulting in additional computational load. This performance constraint can be avoided by setting the [allowSinglePage](#) property to **true** within the [virtualScrollSettings](#).

Enabling this property causes the pivot table to render only the rows and columns that are relevant to the current view page during virtual scrolling. This optimization significantly improves the performance of the pivot table during initial rendering and when performing UI actions such as drill up/down, sorting, filtering, and more.

INDEX.JSX

```
import { PivotViewComponent, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function data(count) {
    let result = [];
    let dt = 0;
    for (let i = 1; i < (count + 1); i++) {
        dt++;
        let round;
        let toString = i.toString();
        if (toString.length === 1) {
            round = '0000' + (i);
        }
        else if (toString.length === 2) {
            round = '000' + i;
        }
        else if (toString.length === 3) {
            round = '00' + i;
        }
        else if (toString.length === 4) {
            round = '0' + i;
        }
        else {
            round = toString;
        }
        result.push({
            ProductID: 'PRO-' + round,
            Year: "FY " + (dt + 2013),
            Price: Math.round(Math.random() * 5000) + 5000,
            Sold: Math.round(Math.random() * 80) + 10,
        });
        if (dt / 4 == 1) {
            dt = 0;
        }
    }
    return result;
};
function App() {
    var pivotObj;
    var virtualScrollSettings = {
```

```

        allowSinglePage: true
    };
    var dataSourceSettings = {
        dataSource: data(1000),
        enableSorting: false,
        expandAll: true,
        formatSettings: [{ name: 'Price', format: 'C0' }],
        rows: [{ name: 'ProductID' }],
        columns: [{ name: 'Year' }],
        values: [{ name: 'Price', caption: 'Unit Price' }, { name: 'Sold',
caption: 'Unit Sold' }]
    };
    return (
        <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
        virtualScrollSettings={virtualScrollSettings}><Inject
services={[GroupingBar]}>
        </PivotViewComponent>
    );
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { PivotViewComponent, IDataOptions, Inject, VirtualScroll,
VirtualScrollSettingsModel } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function data(count: number) {
    let result: Object[] = [];
    let dt: number = 0;
    for (let i: number = 1; i < (count + 1); i++) {
        dt++;
        let round: string;
        let toString: string = i.toString();
        if (toString.length === 1) {
            round = '0000' + (i);
        }
        else if (toString.length === 2) {
            round = '000' + i;
        }
        else if (toString.length === 3) {
            round = '00' + i;
        }
        else if (toString.length === 4) {
            round = '0' + i;
        }
        else {
            round = toString;
        }
        result.push({
            ProductID: 'PRO-' + round,
            Year: "FY " + (dt + 2013),
            Price: Math.round(Math.random() * 5000) + 5000,
            Sold: Math.round(Math.random() * 80) + 10,
        });
        if (dt / 4 == 1) {

```

```

        dt = 0;
    }
}
return result;
};
let virtualScrollSettings: VirtualScrollSettingsModel = {
    allowSinglePage: true
} as VirtualScrollSettingsModel;
function App() {
    let pivotObj: PivotViewComponent;
    let dataSourceSettings: IDataOptions = {
        dataSource: data(1000),
        enableSorting: false,
        expandAll: true,
        formatSettings: [{ name: 'Price', format: 'C0' }],
        rows: [{ name: 'ProductID' }],
        columns: [{ name: 'Year' }],
        values: [{ name: 'Price', caption: 'Unit Price' }, { name: 'Sold',
caption: 'Unit Sold' }]
    }
    return (
        <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
        virtualScrollSettings={virtualScrollSettings}><Inject
services={[VirtualScroll]}/>
        </PivotViewComponent>
    );
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Limitations for virtual scrolling

- In virtual scrolling, the [columnWidth](#) property in [gridSettings](#) should be in pixels, and percentage values are not accepted.
- Resizing columns or setting width to individual columns affects the calculation used to pick the correct page on scrolling.
- Grouping, which takes additional time to splitting the raw items into the provided format.
- Date Formatting, which takes additional time to convert date format.
- Date Formatting with sorting, here additionally full date time format should be framed to perform sorting along with the provided date format which lags the performance.
- When using OLAP data, subtotals and grand totals are only displayed when measures are bound at the last position in the [rows](#) or [columns](#) axis. Otherwise, the data from the pivot table will be shown without summary totals.
- Even if virtual scrolling is enabled, not only is the current view port data retrieved, but also the data for the immediate previous page and the immediate next page. As a result, when the end user scrolls slightly ahead or behind, the next or previous page data is displayed immediately without requiring a refresh. **Note:** If the pivot table's width and height are large, the loading data count in the current, previous, and next viewports (pages) will also increase, affecting performance.

Virtual scrolling for static field list

Virtual scrolling automatically works with "Popup" field list on setting the [enableVirtualization](#) property in the Pivot Table to **true**. In case of static field list, which act as a separate component, user need to enable [enableVirtualization](#) property in the Pivot Table and also pass the report information to pivot table instance via the [load](#) event of the field list.

INDEX.JSX

```
import { CalculatedField, PivotFieldListComponent, Inject,
PivotViewComponent, VirtualScroll } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
const SAMPLE_CSS = `
.e-pivotview {
    width: 58%;
    height: 100%;
    float: left;
}
.e-pivotfieldlist {
    width: 42%;
    height: 100%;
    float: right;
}
.e-pivotfieldlist .e-static {
    width: 100% !important;
}
`;
function data(count) {
    let result = [];
    let dt = 0;
    for (let i = 1; i < (count + 1); i++) {
        dt++;
        let round;
        let toString = i.toString();
        if (toString.length === 1) {
            round = '0000' + (i);
        }
        else if (toString.length === 2) {
            round = '000' + i;
        }
        else if (toString.length === 3) {
            round = '00' + i;
        }
        else if (toString.length === 4) {
            round = '0' + i;
        }
        else {
            round = toString;
        }
        result.push({
            ProductID: 'PRO-' + (i % 1000),
            Year: "FY " + (dt + 2013),
            Price: Math.round(Math.random() * 5000) + 5000,
            Sold: Math.round(Math.random() * 80) + 10,
        });
        if (dt / 4 === 1) {
            dt = 0;
        }
    }
}
```

```

    }
  }
  return result;
}
;
function App() {
  let pivotGridModule;
  let dataSourceSettings = {
    dataSource: data(1000),
    enableSorting: false,
    expandAll: true,
    formatSettings: [{ name: 'Price', format: 'C0' }],
    rows: [{ name: 'ProductID' }],
    columns: [{ name: 'Year' }],
    values: [{ name: 'Price', caption: 'Unit Price' }, { name: 'Sold',
caption: 'Unit Sold' }]
  };
  let pivotObj;
  let fieldListObj;
  return (<div className="control-section">
    <style>{SAMPLE_CSS}</style>
    <PivotViewComponent id='PivotView' ref={d => pivotObj = d}
    enableVirtualization={true} enginePopulated={afterPivotPopulate.bind(this)}
    width={'99%'} height={'530'}><Inject
    services={[VirtualScroll]}></PivotViewComponent>
    <PivotFieldListComponent id='PivotFieldList' ref={d => fieldListObj = d}
    load={onLoad} enginePopulated={afterPopulate.bind(this)}
    dataSourceSettings={dataSourceSettings} renderMode={"Fixed"}
    allowCalculatedField={true}><Inject
    services={[CalculatedField]}></PivotFieldListComponent></div>);
  function afterPopulate() {
    pivotObj = document.getElementById('PivotView').ej2_instances[0];
    fieldListObj =
document.getElementById('PivotFieldList').ej2_instances[0];
    fieldListObj.updateView(pivotObj);
  }
  function afterPivotPopulate() {
    pivotObj = document.getElementById('PivotView').ej2_instances[0];
    fieldListObj =
document.getElementById('PivotFieldList').ej2_instances[0];
    fieldListObj.update(pivotObj);
  }
  function rendereComplete() {
    fieldListObj.updateView(pivotObj);
    fieldListObj.update(pivotObj);
  }
  function onLoad() {
    //Getting component instance.
    pivotObj = document.getElementById('PivotView').ej2_instances[0];
    fieldListObj =
document.getElementById('PivotFieldList').ej2_instances[0];
    pivotGridModule = pivotObj;
    //Assigning report to pivot table component.
    pivotObj.dataSourceSettings = fieldListObj.dataSourceSettings;
    //Generating page settings based on pivot table component's size.
    pivotObj.updatePageSettings(true);
    //Assigning page settings to field list component.
  }
}

```



```

        fieldListObj.pageSettings = pivotObj.pageSettings;
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { CalculatedField, PivotFieldListComponent, IDataOptions, Inject,
PivotViewComponent, VirtualScroll } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
const SAMPLE_CSS = `
.e-pivotview {
    width: 58%;
    height: 100%;
    float: left;
}
.e-pivotfieldlist {
    width: 42%;
    height: 100%;
    float: right;
}
.e-pivotfieldlist .e-static {
    width: 100% !important;
} `;
function data(count: number) {
    let result: Object[] = [];
    let dt: number = 0;
    for (let i: number = 1; i < (count + 1); i++) {
        dt++;
        let round: string;
        let toString: string = i.toString();
        if (toString.length === 1) {
            round = '0000' + (i);
        }
        else if (toString.length === 2) {
            round = '000' + i;
        }
        else if (toString.length === 3) {
            round = '00' + i;
        }
        else if (toString.length === 4) {
            round = '0' + i;
        }
        else {
            round = toString;
        }
        result.push({
            ProductID: 'PRO-' + (i % 1000),
            Year: "FY " + (dt + 2013),
            Price: Math.round(Math.random() * 5000) + 5000,
            Sold: Math.round(Math.random() * 80) + 10,
        });
        if (dt / 4 === 1) {
            dt = 0;
        }
    }
}

```

```

    }
    return result;
};
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: data(1000),
        enableSorting: false,
        expandAll: true,
        formatSettings: [{ name: 'Price', format: 'C0' }],
        rows: [{ name: 'ProductID' }],
        columns: [{ name: 'Year' }],
        values: [{ name: 'Price', caption: 'Unit Price' }, { name: 'Sold',
caption: 'Unit Sold' }]
    }
    let pivotObj: PivotViewComponent;
    let fieldListObj: PivotFieldListComponent;

    return (<div className="control-section">
        <style>{SAMPLE_CSS}</style>
        <PivotViewComponent id='PivotView' ref={d => pivotObj = d}
enableVirtualization={true} enginePopulated={afterPivotPopulate.bind(this)}
width={'99%'} height={'530'}><Inject
services={[VirtualScroll]}></PivotViewComponent>
        <PivotFieldListComponent id='PivotFieldList' ref={d => fieldListObj = d}
load={onLoad} enginePopulated={afterPopulate.bind(this)}
dataSourceSettings={dataSourceSettings} renderMode={"Fixed"}
allowCalculatedField={true}><Inject services={[CalculatedField]}
/></PivotFieldListComponent></div>);
    function afterPopulate(): void {
        pivotObj = document.getElementById('PivotView').ej2_instances[0];
        fieldListObj =
document.getElementById('PivotFieldList').ej2_instances[0];
        fieldListObj.updateView(pivotObj);
    }
    function afterPivotPopulate(): void {
        pivotObj = document.getElementById('PivotView').ej2_instances[0];
        fieldListObj =
document.getElementById('PivotFieldList').ej2_instances[0];
        fieldListObj.update(pivotObj);
    }
    function rendereComplete(): void {
        fieldListObj.updateView(pivotObj);
        fieldListObj.update(pivotObj);
    }
    function onLoad(): void {
        //Getting component instance.
        pivotObj = document.getElementById('PivotView').ej2_instances[0];
        fieldListObj =
document.getElementById('PivotFieldList').ej2_instances[0];
        fieldListObj.pivotGridModule = pivotObj;
        //Assigning report to pivot table component.
        pivotObj.dataSourceSettings = fieldListObj.dataSourceSettings;
        //Generating page settings based on pivot table component's size.
        pivotObj.updatePageSettings(true);
        //Assigning page settings to field list component.
        fieldListObj.pageSettings = pivotObj.pageSettings;
    }
}

```

```
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

See also

- [Paging in Syncfusion EJ2 Typescript components](#)
- [Data Compression in Syncfusion EJ2 Typescript components](#)

Paging in React Pivotview component

Paging allows you to load large amounts of data that can be divided and displayed page by page in the pivot table. It can be enabled by setting the [enablePaging](#) property to **true**. It can be configured at code-behind by using the [pageSettings](#) property, during initial rendering of the component. The properties required are:

- [currentRowPage](#): Allows user to set the current row page number to be displayed in the pivot table.
- [currentColumnPage](#): Allows user to set the current column page number to be displayed in the pivot table.
- [rowPageSize](#): Allows user to set the total number of records to be displayed on each page of the pivot table's row axis.
- [columnPageSize](#): Allows user to set the total number of records to be displayed on each page of the pivot table's column axis.

Pager UI

When paging is enabled, a built-in pager UI appears at the bottom of the pivot table, allowing you to change the current page in the row and column axes by navigating to a desired page using the navigation buttons or an input text box, as well as change the page size via dropdown at runtime.

You can also change the position, visibility, compact view, and template of the row and column pagers by using the [pagerSettings](#).

In order to see and use the pager UI, insert the **Pager** module into the pivot table using the **services** tag.

INDEX.JSX

```
{% raw %}
import { PivotViewComponent, Pager, Inject } from '@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj;
    let remoteData = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings = {
        type: 'JSON',
        dataSource: remoteData,
```

```

        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
        filters: []
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
    rowPageSize: 10,
    columnPageSize: 5,
    currentColumnPage: 1,
    currentRowPage: 1
}} pagerSettings={{
    position: 'Bottom',
    enableCompactView: false,
    showColumnPager: true,
    showRowPager: true,
    columnPageSizes: [5, 10, 20, 50, 100],
    rowPageSizes: [10, 50, 100, 200],
    isInversed: false,
    showColumnPageSize: true,
    showRowPageSize: true
}} enablePaging={true}>
        <Inject services={[Pager]}/>
    </PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { IDataOptions, PivotViewComponent, Pager, Inject } from
'@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj: PivotViewComponent;
    let remoteData: DataManager = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings: IDataOptions = {
        type: 'JSON',
        dataSource: remoteData,
        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],

```

```

        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
        filters: []
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
    rowPageSize: 10,
    columnPageSize: 5,
    currentColumnPage: 1,
    currentRowPage: 1
}} pagerSettings={{
    position: 'Bottom',
    enableCompactView: false,
    showColumnPager: true,
    showRowPager: true,
    columnPageSizes: [5, 10, 20, 50, 100],
    rowPageSizes: [10, 50, 100, 200],
    isInversed: false,
    showColumnPageSize: true,
    showRowPageSize: true
}} enablePaging={true}>
    <Inject services={[Pager]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

Show pager UI at top or bottom

You can display the pager UI at top or bottom of the pivot table by using the [position](#) property. To show the pager UI at top of the pivot table, set the [position](#) property in [pagerSettings](#) to **Top**.

By default, the pager UI appears at the bottom of the pivot table.

INDEX.JSX

```

{% raw %}
import { PivotViewComponent, Pager, Inject } from '@syncfusion/ej2-react-
pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj;
    let remoteData = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings = {
        type: 'JSON',
        dataSource: remoteData,

```

```

        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
        filters: []
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
    rowPageSize: 10,
    columnPageSize: 5,
    currentColumnPage: 1,
    currentRowPage: 1
}} pagerSettings={{
    position: 'Top'
}} enablePaging={true}>
        <Inject services={[Pager]}/>
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { IDataOptions, PivotViewComponent, Pager, Inject } from
'@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj: PivotViewComponent;
    let remoteData: DataManager = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings: IDataOptions = {
        type: 'JSON',
        dataSource: remoteData,
        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
        filters: []
    };
}

```

```

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
    rowPageSize: 10,
    columnPageSize: 5,
    currentColumnPage: 1,
    currentRowPage: 1
}} pagerSettings={{
    position: 'Top'
}} enablePaging={true}>
    <Inject services={[Pager]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

Inverse pager

Toggles and displays row and column pager. To show the column pager on the left side of the pager UI, set the [isInversed](#) property in [pagerSettings](#) to true.

By default, the row pager is displayed on the left side of the pager UI, while the column pager is displayed on the right side.

INDEX.JSX

```

{% raw %}
import { PivotViewComponent, Pager, Inject } from '@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj;
    let remoteData = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings = {
        type: 'JSON',
        dataSource: remoteData,
        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
        filters: []
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
    rowPageSize: 10,
    columnPageSize: 5,

```

```

        currentColumnPage: 1,
        currentRowPage: 1
    }} pagerSettings={{
        isInversed: true
    }} enablePaging={true}>
        <Inject services={[Pager]}/>
    </PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { IDataOptions, PivotViewComponent, Pager, Inject } from
'@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj: PivotViewComponent;
    let remoteData: DataManager = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings: IDataOptions = {
        type: 'JSON',
        dataSource: remoteData,
        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
        filters: []
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pagerSettings={{
        rowPageSize: 10,
        columnPageSize: 5,
        currentColumnPage: 1,
        currentRowPage: 1
    }} pagerSettings={{
        isInversed: true
    }} enablePaging={true}>
        <Inject services={[Pager]} />
    </PivotViewComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```


Compact view

By hiding all except the previous and next navigation buttons, the pager UI can be displayed with the absolute minimum of paging options. The compact view can be enabled by setting the [enableCompactView](#) property in [pagerSettings](#) to **true**.

INDEX.JSX

```
{% raw %}
import { PivotViewComponent, Pager, Inject } from '@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj;
    let remoteData = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings = {
        type: 'JSON',
        dataSource: remoteData,
        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name: 'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit Price' }],
        filters: []
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
        columnWidth: 120 }} pagerSettings={{
        rowPageSize: 10,
        columnPageSize: 5,
        currentColumnPage: 1,
        currentRowPage: 1
    }} pagerSettings={{
        enableCompactView: true
    }} enablePaging={true}>
        <Inject services={[Pager]}/>
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
```

```

import { IDataOptions, PivotViewComponent, Pager, Inject } from
 '@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj: PivotViewComponent;
    let remoteData: DataManager = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings: IDataOptions = {
        type: 'JSON',
        dataSource: remoteData,
        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
        filters: []
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pagerSettings={{
    rowPageSize: 10,
    columnPageSize: 5,
    currentColumnPage: 1,
    currentRowPage: 1
}} pagerSettings={{
    enableCompactView: true
}} enablePaging={true}>
        <Inject services={[Pager]} />
    </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

Show or hide paging option

By using the [showRowPager](#) and [showColumnPager](#) properties in [pagerSettings](#), you can show or hide row and column pager separately in the pager UI.

In the following example, row pager has been disabled by setting the [showRowPager](#) property in [pagerSettings](#) to **false**.

INDEX.JSX

```

{% raw %}
import { PivotViewComponent, Pager, Inject } from '@syncfusion/ej2-react-
pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```
function App() {
  let pivotObj;
  let remoteData = new DataManager({
    url: 'https://bi.syncfusion.com/northwindservice/api/orders',
    adaptor: new WebApiAdaptor,
    crossDomain: true
  });
  let dataSourceSettings = {
    type: 'JSON',
    dataSource: remoteData,
    expandAll: true,
    columns: [{ name: 'ProductName', caption: 'Product Name' }],
    rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
    formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
    values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
    filters: []
  };
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
  rowPageSize: 10,
  columnPageSize: 5,
  currentColumnPage: 1,
  currentRowPage: 1
}} pagerSettings={{
  showRowPager: false
}} enablePaging={true}>
    <Inject services={[Pager]}/>
  </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import { IDataOptions, PivotViewComponent, Pager, Inject } from
'@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let pivotObj: PivotViewComponent;
  let remoteData: DataManager = new DataManager({
    url: 'https://bi.syncfusion.com/northwindservice/api/orders',
    adaptor: new WebApiAdaptor,
    crossDomain: true
  });
  let dataSourceSettings: IDataOptions = {
    type: 'JSON',
    dataSource: remoteData,
    expandAll: true,

```

```

        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
        filters: []
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
    rowPageSize: 10,
    columnPageSize: 5,
    currentColumnPage: 1,
    currentRowPage: 1
}} pagerSettings={{
    showRowPager: false
}} enablePaging={true}>
        <Inject services={[Pager]} />
    </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

Show or hide page size

By using the [showRowPageSize](#) and [showColumnPageSize](#) properties in [pagerSettings](#), you can show or hide "Rows per page" and "Columns per page" dropdown menu. The dropdown menu contains a list of pre-defined or user-defined page sizes, which will be displayed in the "Rows per page" and "Columns per page" dropdowns, allowing you to change the page size for the row and column axes at runtime.

INDEX.JSX

```

{% raw %}
import { PivotViewComponent, Pager, Inject } from '@syncfusion/ej2-react-
pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj;
    let remoteData = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings = {
        type: 'JSON',
        dataSource: remoteData,
        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],

```

```

        filters: []
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
    rowPageSize: 10,
    columnPageSize: 5,
    currentColumnPage: 1,
    currentRowPage: 1
}} pagerSettings={{
    showColumnPageSize: false,
    showRowPageSize: false
}} enablePaging={true}>
        <Inject services={[Pager]}/>
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { IDataOptions, PivotViewComponent, Pager, Inject } from
'@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj: PivotViewComponent;
    let remoteData: DataManager = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings: IDataOptions = {
        type: 'JSON',
        dataSource: remoteData,
        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
        filters: []
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
    rowPageSize: 10,
    columnPageSize: 5,
    currentColumnPage: 1,
    currentRowPage: 1
}} pagerSettings={{

```

```

        showColumnPageSize: false,
        showRowPageSize: false
    }} enablePaging={true}>
        <Inject services={[Pager]} />
    </PivotViewComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

Customize page size

By using the [rowPageSizes](#) and [columnPageSizes](#) properties in [pagerSettings](#), you can specify a set of desired page sizes, which will be displayed in the "Rows per page" and "Columns per page" dropdowns, allowing you to change the page size for the row and column axes at runtime.

By default, the "Rows per page" dropdown have pre-defined page sizes of **10, 50, 100, and 200**, while the "Columns per page" dropdown have pre-defined page sizes of **5, 10, 20, 50, and 100**.

In the following example, the "Rows per page" dropdown is set with user-defined page sizes of **10, 20, 30, 40, and 50** and the "Columns per page" dropdown is set with user-defined page sizes of **5, 10, 15, 20, and 30**.

INDEX.JSX

```

{% raw %}
import { PivotViewComponent, Pager, Inject } from '@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj;
    let remoteData = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings = {
        type: 'JSON',
        dataSource: remoteData,
        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name: 'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit Price' }],
        filters: []
    };
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} gridSettings={{ columnWidth: 120 }} pagerSettings={{ rowPageSize: 10, columnPageSize: 5, currentColumnPage: 1, currentRowPage: 1

```

```

    }} pagerSettings={{
      columnPageSizes: [5, 10, 15, 20, 30],
      rowPageSizes: [10, 20, 30, 40, 50]
    }} enablePaging={true}>
    <Inject services={[Pager]}/>
  </PivotViewComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { IDataOptions, PivotViewComponent, Pager, Inject } from
 '@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let pivotObj: PivotViewComponent;
  let remoteData: DataManager = new DataManager({
    url: 'https://bi.syncfusion.com/northwindservice/api/orders',
    adaptor: new WebApiAdaptor,
    crossDomain: true
  });
  let dataSourceSettings: IDataOptions = {
    type: 'JSON',
    dataSource: remoteData,
    expandAll: true,
    columns: [{ name: 'ProductName', caption: 'Product Name' }],
    rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
    formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
    values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
    filters: []
  };
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
  rowPageSize: 10,
  columnPageSize: 5,
  currentColumnPage: 1,
  currentRowPage: 1
}} pagerSettings={{
  columnPageSizes: [5, 10, 15, 20, 30],
  rowPageSizes: [10, 20, 30, 40, 50]
}} enablePaging={true}>
    <Inject services={[Pager]}/>
  </PivotViewComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

Template

The [template](#) property allows to change the appearance of the pager UI by displaying user-defined HTML elements instead of built-in HTML elements.

INDEX.JSX

```
{% raw %}
import { PivotViewComponent, Pager, Inject } from '@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { Pager as GridPager } from '@syncfusion/ej2-grids';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj;
    let remoteData = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings = {
        type: 'JSON',
        dataSource: remoteData,
        expandAll: true,
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
        filters: []
    };
    let rowPager;
    let columnPager;
    function dataBound() {
        updateTemplate();
    }
    function updateTemplate() {
        if (!isNullOrUndefined(rowPager)) {
            rowPager.destroy();
            rowPager = null;
        }
        rowPager = new GridPager({
            pageSize: pivotObj.pageSettings.rowPageSize,
            totalRecordsCount: pivotObj.engineModule.rowCount,
            currentPage: pivotObj.pageSettings.currentRowPage,
            pageCount: 5,
            click: rowPageClick
        });
        rowPager.appendTo('#row-pager');
        if (!isNullOrUndefined(columnPager)) {
            columnPager.destroy();
            columnPager = null;
        }
    }
}
```



```

        columnPager = new GridPager({
            pageSize: pivotObj.pageSettings.columnPageSize,
            totalRecordsCount: pivotObj.engineModule.columnCount,
            currentPage: pivotObj.pageSettings.currentColumnPage,
            pageCount: 5,
            click: columnPageClick
        });
        columnPager.appendTo('#column-pager');
    }
    function rowPageClick(args) {
        pivotObj.pageSettings.currentRowPage = args.currentPage;
        pivotObj.refreshData();
    }
    function columnPageClick(args) {
        pivotObj.pageSettings.currentColumnPage = args.currentPage;
        pivotObj.refreshData();
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
    rowPageSize: 10,
    columnPageSize: 5,
    currentColumnPage: 1,
    currentRowPage: 1
}} pagerSettings={{
    template: '#template'
}} enablePaging={true} dataBound={dataBound.bind(this)}>
        <Inject services={[Pager]}/>
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { IDataOptions, PivotViewComponent, Pager, Inject } from
'syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { Pager as GridPager } from '@syncfusion/ej2-grids';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let pivotObj: PivotViewComponent;
    let remoteData: DataManager = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    let dataSourceSettings: IDataOptions = {
        type: 'JSON',
        dataSource: remoteData,
        expandAll: true,

```

```

        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }],
        filters: []
    };
    let rowPager: GridPager;
    let columnPager: GridPager;
    function dataBound() {
        updateTemplate();
    }
    function updateTemplate() {
        if (!isNullOrUndefined(rowPager)) {
            rowPager.destroy();
            rowPager = null;
        }
        rowPager = new GridPager({
            pageSize: pivotObj.pageSettings.rowPageSize,
            totalRecordsCount: pivotObj.engineModule.rowCount,
            currentPage: pivotObj.pageSettings.currentRowPage,
            pageCount: 5,
            click: rowPageClick
        });
        rowPager.appendTo('#row-pager');
        if (!isNullOrUndefined(columnPager)) {
            columnPager.destroy();
            columnPager = null;
        }
        columnPager = new GridPager({
            pageSize: pivotObj.pageSettings.columnPageSize,
            totalRecordsCount: pivotObj.engineModule.columnCount,
            currentPage: pivotObj.pageSettings.currentColumnPage,
            pageCount: 5,
            click: columnPageClick
        });
        columnPager.appendTo('#column-pager');
    }
    function rowPageClick(args: any) {
        pivotObj.pageSettings.currentRowPage = args.currentPage;
        pivotObj.refreshData();
    }
    function columnPageClick(args: any) {
        pivotObj.pageSettings.currentColumnPage = args.currentPage;
        pivotObj.refreshData();
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} gridSettings={{
columnWidth: 120 }} pageSettings={{
    rowPageSize: 10,
    columnPageSize: 5,
    currentColumnPage: 1,
    currentRowPage: 1
}} pagerSettings={{
    template: '#template'
}} enablePaging={true} dataBound={dataBound.bind(this)}>

```

```

        <Inject services={[Pager]} />
      </PivotViewComponent>;
    };
    export default App;
    ReactDOM.render(<App />, document.getElementById('sample'));
    {% enddraw %}

```

<!-- markdownlint-disable MD036 -->

Data Compression in React Pivot Table component

This property is applicable only for relational data source.

When binding one million raw data, the pivot table processes all raw data to generate aggregated data during initial rendering and report manipulation. However, with data compression, the input raw data is compressed based on the uniqueness of the raw data, and the final compressed raw data are utilized by the pivot table. The compressed raw data is then used for further operations at all times, reducing the looping complexity and improving the performance of the pivot table. For example, if the pivot table is connected to one million raw data compressed to 1,000 unique raw data, it will render within 3 seconds rather than 10 seconds. You can enable this option by using the [allowDataCompression](#) property along with the [enableVirtualization](#) property.

This options will only function when the virtual scrolling is enabled.

INDEX.JSX

```

import { PivotViewComponent, VirtualScroll, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
let date1;
let date2;
function data(count) {
    let result = [];
    let dt = 0;
    for (let i = 1; i < (count + 1); i++) {
        dt++;
        let round;
        let toString = i.toString();
        if (toString.length === 1) {
            round = '0000' + (i);
        }
        else if (toString.length === 2) {
            round = '000' + i;
        }
        else if (toString.length === 3) {
            round = '00' + i;
        }
        else if (toString.length === 4) {
            round = '0' + i;
        }
        else {
            round = toString;
        }
        result.push({
            ProductID: 'PRO-' + (i % 1000),

```

```

        Year: "FY " + (dt + 2013),
        Price: Math.round(Math.random() * 5000) + 5000,
        Sold: Math.round(Math.random() * 80) + 10,
    });
    if (dt / 4 == 1) {
        dt = 0;
    }
}
return result;
}
;
function App() {
    let dataSourceSettings = {
        dataSource: data(1000),
        enableSorting: false,
        expandAll: true,
        formatSettings: [{ name: 'Price', format: 'C0' }],
        rows: [{ name: 'ProductID' }],
        columns: [{ name: 'Year' }],
        values: [{ name: 'Price', caption: 'Unit Price' }, { name: 'Sold',
caption: 'Unit Sold' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enableVirtualization={true} allowDataCompression={true}
dataSourceSettings={dataSourceSettings}><Inject
services={[VirtualScroll]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, VirtualScroll, Inject }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
let datel: number;
let date2: number;
function data(count: number) {
    let result: Object[] = [];
    let dt: number = 0;
    for (let i: number = 1; i < (count + 1); i++) {
        dt++;
        let round: string;
        let toString: string = i.toString();
        if (toString.length === 1) {
            round = '0000' + (i);
        }
        else if (toString.length === 2) {
            round = '000' + i;
        }
        else if (toString.length === 3) {
            round = '00' + i;
        }
        else if (toString.length === 4) {

```

```

        round = '0' + i;
    } else {
        round = toString;
    }
    result.push({
        ProductID: 'PRO-' + (i % 1000),
        Year: "FY " + (dt + 2013),
        Price: Math.round(Math.random() * 5000) + 5000,
        Sold: Math.round(Math.random() * 80) + 10,
    });
    if (dt / 4 == 1) {
        dt = 0;
    }
}
return result;
};
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: data(1000),
        enableSorting: false,
        expandAll: true,
        formatSettings: [{ name: 'Price', format: 'C0' }],
        rows: [{ name: 'ProductID' }],
        columns: [{ name: 'Year' }],
        values: [{ name: 'Price', caption: 'Unit Price' }, { name: 'Sold',
caption: 'Unit Sold' }]
    }
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enableVirtualization={true} allowDataCompression={true}
dataSourceSettings={dataSourceSettings}><Inject
services={[VirtualScroll]}/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Limitations during data compression:

- The following aggregation types will not be supported:
- Average
- Populationstdev
- Samplestdev
- Populationvar
- Samplevar
- If you use any of the above aggregations, they will result in the aggregation type **“Sum”**.
- **“DistinctCount”** will act as **“Count”** aggregation type.
- In the calculated field, an existing field can be inserted without altering its default aggregation type. Even if changed, it would revert to the default aggregation type for calculation.

State persistence in React Pivotview component

State persistence allows user to maintain the current state of the component along with its report bounded in the browser local storage (cookie). Even if the browser is refreshed or if you move to the

next page within the browser, components state will be persisted. State persistence stores the Pivot Table object in the local storage when [enablePersistence](#) property is set to **true**.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        dataSource: pivotData,
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France', 'Germany'] }],
        columns: [{ name: 'Year' }, { name: 'Order_Source', caption: 'Order
Source' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'In_Stock', caption: 'In Stock' },
            { name: 'Sold', caption: 'Units Sold' }],
        filters: [{ name: 'Product_Categories', caption: 'Product
Categories' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enablePersistence={true}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData,
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France', 'Germany'] }],
        columns: [{ name: 'Year' }, { name: 'Order_Source', caption: 'Order
Source' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'In_Stock', caption: 'In Stock' },
            { name: 'Sold', caption: 'Units Sold' }],
        filters: [{ name: 'Product_Categories', caption: 'Product
Categories' }]
    }
    let pivotObj: PivotViewComponent;
```

```

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enablePersistence={true}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Save and Load Pivot Layout

You can save the current layout of the pivot table by using [getPersistData](#) in string format. The saved layout can be loaded to pivot table any time by passing the saved data as a parameter to [Link to the Video](#) method.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        dataSource: pivotData,
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France', 'Germany'] }],
        columns: [{ name: 'Year' }, { name: 'Order_Source', caption: 'Order
Source' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'In_Stock', caption: 'In Stock' },
            { name: 'Sold', caption: 'Units Sold' }],
        filters: [{ name: 'Product_Categories', caption: 'Product
Categories' }]
    };
    let pivotObj;
    let report;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}></PivotViewComponent></div><div
className='col-lg-3 property-section'><div><ButtonComponent cssClass='e-
primary' onClick={save.bind(this)}>Save</ButtonComponent></div><br
/><div><ButtonComponent cssClass='e-primary'
onClick={load.bind(this)}>Load</ButtonComponent></div></div>
</div>);
    function save() {
        report = pivotObj.getPersistData();
    }
    function load() {
        pivotObj.loadPersistData(report);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, Inject, PivotViewComponent } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData,
    expandAll: false,
    enableSorting: true,
    drilledMembers: [{ name: 'Country', items: ['France', 'Germany'] }],
    columns: [{ name: 'Year' }, { name: 'Order_Source', caption: 'Order
Source' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'In_Stock', caption: 'In Stock' },
    { name: 'Sold', caption: 'Units Sold' }],
    filters: [{ name: 'Product_Categories', caption: 'Product
Categories' }]
  }
  let pivotObj: PivotViewComponent;
  let report: string;
  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}></PivotViewComponent></div><div
className='col-lg-3 property-section'><div><ButtonComponent cssClass='e-
primary'
onClick={save.bind(this)}>Save</ButtonComponent></div><br/><div><ButtonCompo
nent cssClass='e-primary'
onClick={load.bind(this)}>Load</ButtonComponent></div></div>
</div>);
  function save(): void {
    report = pivotObj.getPersistData();
  }
  function load(): void {
    pivotObj.loadPersistData(report);
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Row and column in React Pivotview component

To learn about how to use the row and column options effectively in the React Pivot Table, watch this video:

Width And Height

Allows end user to set the pivot table's height and width by using [height](#) and [width](#) properties respectively. The supported formats to set [height](#) and [width](#) properties are,

- Pixel: For example - 100, 200, "100px", "200px".
- Percentage: For example - "100%", "200%".
- Auto: It is applicable for [height](#) property alone in-order to render the pivot table beyond its parent container height without vertical scrollbar. The parent container here would show its vertical scrollbar as soon as the component reaches beyond its dimension.

The pivot table will not be displayed less than **400px**, since it's the minimum width of the component.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} width={'100%'}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataSet, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataSet[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} width={'100%'}dataSourceSettings={dataSourceSettings}
></PivotViewComponent>);
};
```

```
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Row Height

Allows end user to set the height of each pivot table rows commonly using the [rowHeight](#) property in [gridSettings](#).

By default, the [rowHeight](#) property is set as **36** pixels for desktop layout and **48** pixels for mobile layout.

The height of the column headers alone may vary when grouping bar feature is enabled.

In the below code sample, the [rowHeight](#) property is set as **60** pixels.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        rowHeight: 60
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
function App() {
    let gridSettings: GridSettings = {
        rowHeight: 60
    } as GridSettings;
```

```

let dataSourceSettings: IDataOptions = {
  columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
  dataSource: pivotData as IDataset[],
  expandAll: false,
  filters: [],
  formatSettings: [{ name: 'Amount', format: 'C0' }],
  rows: [{ name: 'Country' }, { name: 'Products' }],
  values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
}
let pivotObj: PivotViewComponent;

return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Column Width

Allows end user to set the width of each pivot table columns commonly using the [columnWidth](#) property in [gridSettings](#).

By default, the [columnWidth](#) property is set as **110** pixels to each columns except the first column. For first column, **250** pixels and **200** pixels are set respectively with and without grouping bar.

In the below example, the [columnWidth](#) property is set as **200** pixels.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let gridSettings = {
    columnWidth: 120
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}

```

```
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
    let gridSettings: GridSettings = {
        columnWidth: 120
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    };
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} gridSettings={gridSettings} dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Adjust width based on columns

By default, if the component width set in code-behind is more than the width of the total columns, then the columns will be stretched to make it fit. To avoid the stretching, set the [allowAutoResizing](#) property in the [gridSettings](#) to **false**. By doing so, the component will be adjusted (shrunk) based on the width of total columns.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        allowAutoResizing: false
    };
    let dataSourceSettings = {
```

```

        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filterSettings: [{ name: 'Year', type: 'Exclude', items: ['FY 2015',
'FY 2017'] }],
        rows: [{ name: 'Country' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
function App() {
    let gridSettings: GridSettings = {
        allowAutoResizing: false
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filterSettings: [{ name: 'Year', type: 'Exclude', items: ['FY 2015', 'FY
2017'] }],
        rows: [{ name: 'Country' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Reorder

Allows end user to reorder a particular column header from one index to another index within the pivot table through drag-and-drop option. It can be enabled by setting the [allowReordering](#) property in [gridSettings](#) to **true**.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        allowReordering: true
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' } ]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
function App() {
    let gridSettings: GridSettings = {
        allowReordering: true
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
```

```

    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Column Resizing

Allows end user to resize the columns by clicking and dragging the right edge of the column header. While dragging, the width of the respective column will be resized immediately. To enable column resizing option, set the [allowResizing](#) property in [gridSettings](#) to **true**.

By default, the column resizing option is enabled.

In RTL mode, user can click and drag the left edge of the header cell to resize the column.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let gridSettings = {
    allowResizing: true
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
  let gridSettings: GridSettings = {
    allowResizing: true
  } as GridSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

In RTL mode, you can click and drag the left edge of the header cell to resize the column.

Text Wrap

Allows end user to wrap the cell content to the next line when it exceeds the boundary of the cell width. To enable text wrap, set the [allowTextWrap](#) property in [gridSettings](#) to **true**.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let gridSettings = {
    allowTextWrap: true
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
```



```

        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
function App() {
    let gridSettings: GridSettings = {
        allowTextWrap: true
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Text Align

Allows end user to align the content of the pivot table's row and column headers and value cells by using the [textAlign](#) and [headerTextAlign](#) properties in the [columnRender](#) event under [gridSettings](#). The following alignments are:

- **Left** - It allows the content to be positioned on the left.
- **Right** - It allows the content to be positioned on the right.

- **Center** - It allows the content to be positioned in the middle.
- **Justify** - It allows the content to be as flexible as possible, when the cell does not occupy the entire available area.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        columnRender: columnRender.bind(this)
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    function columnRender(args) {
        for (let i = 0; i < args.columns.length; i++) {
            if (args.stackedColumns[0]) {
                // Content for the row headers is right-aligned here.
                args.stackedColumns[0].textAlign = "Right";
            }
            if (args.stackedColumns[1]) {
                // Content for the column header "FY 2015" is center-aligned
here.
                args.stackedColumns[1].textAlign = 'Center';
            }
            if (args.stackedColumns[1] && args.stackedColumns[1].columns[0])
{
                // Content for the column header "Q1" is right-aligned here.
                args.stackedColumns[1].columns[0].textAlign = 'Right';
            }
            if (args.stackedColumns[1] && args.stackedColumns[1].columns[0]
&& args.stackedColumns[1].columns[0].columns[0]) {
                // Content for the value header "Units Sold" is right-
aligned here.
                args.stackedColumns[1].columns[0].columns[0].headerTextAlign
= 'Right';
            }
            if (args.stackedColumns[1] && args.stackedColumns[1].columns[0]
&& args.stackedColumns[1].columns[0].columns[0]) {
                // Content for the values are left-aligned here.
                args.stackedColumns[1].columns[0].columns[0].textAlign =
'Left';
            }
        }
    }
}
```

```

    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
    let gridSettings: GridSettings = {
        columnRender: columnRender.bind(this)
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    function columnRender(args) {
        for (let i: number = 0; i < args.columns.length; i++) {
            if (args.stackedColumns[0]) {
                // Content for the row headers is right-aligned here.
                args.stackedColumns[0].textAlign = "Right";
            }
            if (args.stackedColumns[1]) {
                // Content for the column header "FY 2015" is center-aligned here.
                args.stackedColumns[1].textAlign = 'Center';
            }
            if (args.stackedColumns[1] && (args.stackedColumns[1].columns[0] as
any)) {
                // Content for the column header "Q1" is right-aligned here.
                (args.stackedColumns[1].columns[0] as any).textAlign = 'Right';
            }
            if (args.stackedColumns[1] && args.stackedColumns[1].columns[0] &&
(args.stackedColumns[1].columns[0] as any).columns[0]) {
                // Content for the value header "Units Sold" is right-aligned
here.
                (args.stackedColumns[1].columns[0] as
any).columns[0].headerTextAlign = 'Right';
            }
        }
    }
}

```

```

    }
    if(args.stackedColumns[1] && args.stackedColumns[1].columns[0] &&
    (args.stackedColumns[1].columns[0] as any).columns[0]){
        // Content for the values are left-aligned here.
        (args.stackedColumns[1].columns[0] as any).columns[0].textAlign =
        'Left';
    }
}
}

return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

AutoFit

Allows the user to fit the Pivot Table columns as wide as the content of the cell without wrapping. It auto fits all of the Pivot Table columns by invoking the [autoFitColumns](#) method from the grid instance.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
        'Quarter' }],
        dataSource: pivotData,
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
        caption: 'Sold Amount' } ]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
dataBound={trend.bind(this)}></PivotViewComponent>);
    function trend() {
        pivotObj.grid.autoFitColumns();
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';

```

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: true,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} dataBound={trend.bind(this)}></PivotViewComponent>);
  function trend(): void {
    pivotObj.grid.autoFitColumns();
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

The minimum width of 250 pixels is set by default with the grouping bar UI for the first column and cannot be reduced further. So, when the grouping bar is enabled, one can auto fit the Pivot Table columns by calling the [autoFitColumns](#) method from the grid instance with the parameter contained pivot table columns field name excluding first column.

INDEX.JSX

```

import { GroupingBar, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let groupingSettings = {
    showFieldsPanel: true
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: pivotData,
    expandAll: true,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
  };
};

```

```

    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} groupingBarSettings={groupingSettings}
dataBound={trend.bind(this)} dataSourceSettings={dataSourceSettings}
showGroupingBar={true}><Inject services={[GroupingBar]}/>
</PivotViewComponent>);
    function trend() {
        if (pivotObj.showGroupingBar) {
            let columns = [];
            for (let i = 1; i < pivotObj.grid.columnModel.length; i++) {
                columns.push(pivotObj.grid.columnModel[i].field);
            }
            pivotObj.grid.autoFitColumns(columns);
        }
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { GroupingBar, GroupingBarSettings, IDataOptions, IDataset, Inject,
PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let groupingSettings: GroupingBarSettings = {
        showFieldsPanel: true
    } as GroupingBarSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} groupingBarSettings={groupingSettings}
dataBound={trend.bind(this)} dataSourceSettings={dataSourceSettings}
showGroupingBar={true} ><Inject services={[GroupingBar]}/>
</PivotViewComponent>);
    function trend(): void {
        if (pivotObj.showGroupingBar) {
            let columns: string[] = [];
            for (let i: number = 1; i < (pivotObj.grid as any).columnModel.length;
i++) {
                columns.push((pivotObj.grid as any).columnModel[i].field);
            }
        }
    }
}

```

```

        pivotObj.grid.autoFitColumns(columns);
    }
}
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Autofit specific columns

During initial rendering, the parameter `autoFit` in the `columnRender` event under `gridSettings` can be set to `true` to auto fit specific columns.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        columnRender: columnRender.bind(this)
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    function columnRender(args) {
        for (let i = 0; i < args.columns.length; i++) {
            args.columns[i].autoFit = true;
        }
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';

```

```
function App() {
  let gridSettings: GridSettings = {
    columnRender: columnRender.bind(this)
  } as GridSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: true,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  function columnRender(args) {
    for (let i: number = 0; i < args.columns.length; i++) {
      args.columns[i].autoFit = true;
    }
  }

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Grid Lines

Allows end user to display cell border for each cells using [gridLines](#) property in [gridSettings](#).

Available mode of grid lines are:

- | Modes | Actions |
- |-----|-----|
- | Both | Displays both the horizontal and vertical grid lines.|
- | None | No grid lines are displayed.|
- | Horizontal | Displays the horizontal grid lines only.|
- | Vertical | Displays the vertical grid lines only.|
- | Default | Displays grid lines based on the theme.|

By default, pivot table renders grid lines in **Both** mode.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let gridSettings = {
    gridLines: 'Vertical'
```



```

    };
    let dataSourceSettings = {
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      dataSource: pivotData,
      expandAll: false,
      filters: [],
      drilledMembers: [{ name: 'Country', items: ['France' ]}],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
  }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
function App() {
  let gridSettings: GridSettings = {
    gridLines: 'Vertical'
  } as GridSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Selection

Selection provides an option to highlight a row or a column or a cell. It can be done through simple mouse down or arrow keys. To enable selection in the pivot table, set the [allowSelection](#) property in [gridSettings](#) to **true**.

The pivot table supports two types of selection that can be set using [type](#) property in [selectionSettings](#). The selection types are:

- **Single:** It is set by default, and it only allows selection of a single row or a column or a cell.
- **Multiple:** Allows you to select multiple rows or columns or cells. To perform multi-selection, press and hold "CTRL" key and click the desired rows or cells. To select range of rows or cells, press and hold the "SHIFT" key and click the rows or columns or cells.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        allowSelection: true,
        selectionSettings: { type: 'Multiple' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
```

```
function App() {
  let gridSettings: GridSettings = {
    allowSelection: true,
    selectionSettings: { type: 'Multiple' }
  } as GridSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Selection Mode

The pivot table supports four types of selection mode that can be set using [mode](#) in [selectionSettings](#). The selection modes are:

- **Row:** It is set by default, and allows user to select only rows.
- **Column:** Allows user to select only columns.
- **Cell:** Allows user to select only cells.
- **Both:** Allows user to select rows and columns at the same time.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let gridSettings = {
    allowSelection: true,
    selectionSettings: { mode: 'Both' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
```

```

        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
    let gridSettings: GridSettings = {
        allowSelection: true,
        selectionSettings: { mode: 'Both' }
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Cell Selection Mode

The pivot table supports two types of cell selection mode that can be set using [cellSelectionMode](#) in [selectionSettings](#). The cell selection modes are:

- **Flow:** It is set by default. The range of cells are selected between the start index and end index that includes in-between cells of rows.

- **Box:** Range of cells are selected from the start and end column indexes that includes in-between cells of rows within the range.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        allowSelection: true,
        selectionSettings: { cellSelectionMode: 'Box', type: 'Multiple',
mode: 'Cell' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataSet, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
function App() {
    let gridSettings: GridSettings = {
        allowSelection: true,
        selectionSettings: { cellSelectionMode: 'Box', type: 'Multiple', mode:
'Cell' }
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataSet[],
        expandAll: false,
```

```

    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France'] }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Cell selection requires [mode](#) property in [selectionSettings](#) to be **Cell** or **Both**, and [type](#) property should be **Multiple**.

Changing background color of the selected cell

The background-color of the selected cell can be changed using built-in CSS names. To do so, please refer to the code sample below, which shows that the selected cells are changed to a **green yellow** color.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
const SAMPLE_CSS = `
.e-pivotview .e-cellselectionbackground,
.e-pivotview .e-selectionbackground,
.e-pivotview .e-grid .e-rowsheader .e-selectionbackground,
.e-pivotview .e-grid .e-columnsheader.e-selectionbackground {
  background-color: greenYellow !important;
}
`;
function App() {
  let gridSettings = {
    allowSelection: true,
    selectionSettings: { cellSelectionMode: 'Box', type: 'Multiple',
mode: 'Cell' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France'] }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<div>

```

```

        <style>{SAMPLE_CSS}</style>
        <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>
    </div>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
const SAMPLE_CSS = `
.e-pivotview .e-cellselectionbackground,
.e-pivotview .e-selectionbackground,
.e-pivotview .e-grid .e-rowsheader .e-selectionbackground,
.e-pivotview .e-grid .e-columnsheader .e-selectionbackground {
    background-color: greenYellow !important;
}`;

function App() {
    let gridSettings: GridSettings = {
        allowSelection: true,
        selectionSettings: { cellSelectionMode: 'Box', type: 'Multiple', mode: 'Cell' }
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData as IDataset[],
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (
        <div>
            <style>{SAMPLE_CSS}</style>
            <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>
        </div>
    );
};
export default App;

```

```
ReactDOM.render(<App />, document.getElementById('sample'));
```

Event

CellSelected

The event [cellSelected](#) is triggered when cell selection gets completed. It provides selected cells information with its corresponding column and row headers. It has following parameters - `selectedCellsInfo`, `currentCell` and `target`. This event allows user to view selected cells information and user can pass those selected cells information to any external component for data binding.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        allowSelection: true,
        selectionSettings: { mode: 'Both', type: 'Multiple' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C2', useGrouping: false,
minimumSignificantDigits: 1, maximumSignificantDigits: 3 }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    };
    function onCellSelected(args) {
        //args.selectedCellsInfo -> get selected cells information.
        //args.pivotValues -> get the pivot values of the pivot table.
    }
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings}
cellSelected={onCellSelected}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent, GridSettings,
PivotCellSelectedEventArgs } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
```



```

import { pivotData } from './datasource';
function App() {
  let gridSettings: GridSettings = {
    allowSelection: true,
    selectionSettings: { mode: 'Both', type: 'Multiple' }
  } as GridSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C2', useGrouping: false,
minimumSignificantDigits: 1, maximumSignificantDigits: 3 }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
  }
  function onCellSelected(args: PivotCellSelectedEventArgs): void {
    //args.selectedCellsInfo -> get selected cells information.
    //args.pivotValues -> get the pivot values of the pivot table.
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings}
cellSelected={onCellSelected}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

CellSelecting

The event `cellSelecting` triggers before cell gets selected gets completed. It provides selected cells information with its corresponding column and row headers. It has following parameters - `currentCell`, `data` and `cancel`.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let gridSettings = {
    allowSelection: true,
    selectionSettings: { mode: 'Both', type: 'Multiple' }
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,

```

```

        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C2', useGrouping: false,
minimumSignificantDigits: 1, maximumSignificantDigits: 3 }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    };
    function onCellSelecting(args) {
        //args.selectedCellsInfo -> get selected cells information.
        //args.pivotValues -> get the pivot values of the pivot table.
    }
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings}
cellSelecting={onCellSelecting}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, GridSettings,
PivotCellSelectedEventArgs } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings: GridSettings = {
        allowSelection: true,
        selectionSettings: { mode: 'Both', type: 'Multiple' }
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C2', useGrouping: false,
minimumSignificantDigits: 1, maximumSignificantDigits: 3 }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    }
    function onCellSelecting(args: PivotCellSelectedEventArgs): void {
        //args.selectedCellsInfo -> get selected cells information.
        //args.pivotValues -> get the pivot values of the pivot table.
    }
    let pivotObj: PivotViewComponent;

```

```

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings}
cellSelecting={onCellSelecting}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Clip Mode

The clip mode provides options to display its overflow cell content in the pivot table. It can be configured using the [clipMode](#) property in [gridSettings](#). The pivot table supports three types of clip modes which are:

- **Clip:** Truncates the cell content when it overflows its area.
- **Ellipsis:** Displays ellipsis when the cell content overflows its area.
- **EllipsisWithTooltip:** Displays ellipsis when the cell content overflows its area, also it will display the tooltip while hover on ellipsis is applied.

By default, [clipMode](#) value is set to **Ellipsis**.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        clipMode: 'Clip'
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
  let gridSettings: GridSettings = {
    clipMode: 'Clip'
  } as GridSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
  };
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} gridSettings={gridSettings} dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Cell Template

User can customize the pivot table cell element by using the [cellTemplate](#) property. The [cellTemplate](#) property accepts either an HTML string or the element's ID, which can be used to append additional HTML elements to showcase each cell with custom format.

In this demo, the revenue cost for each year is represented with trend icons.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { renewableEnergy } from './datasource';
function App() {
  function cellTemplate(props) {
    return (<span className="tempwrap sb-icon-neutral pv-icons"></span>);
  }
  let dataSourceSettings = {
    dataSource: renewableEnergy,
    expandAll: true,
    enableSorting: true,
    drilledMembers: [{ name: 'Year', items: ['FY 2015', 'FY 2017', 'FY 2018' ]}],
  };
  let pivotObj: PivotViewComponent;
```

```

formatSettings: [{ name: 'ProCost', format: 'C0' }],
rows: [
  { name: 'Year', caption: 'Production Year' }
],
columns: [
  { name: 'EnerType', caption: 'Energy Type' },
  { name: 'EneSource', caption: 'Energy Source' }
],
values: [
  { name: 'ProCost', caption: 'Revenue Growth' }
],
filters: []
};
let pivotObj;
return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
cellTemplate={cellTemplate.bind(this)}
dataBound={trend.bind(this)}></PivotViewComponent>);
function trend() {
  let cTable = [].slice.call(document.getElementsByClassName("e-
table"));
  let colLen = pivotObj.pivotValues[3].length;
  let cLen = cTable[3].children[0].children.length;
  let rLen = cTable[3].children[1].children.length;
  let rowIndx;
  for (let k = 0; k < rLen; k++) {
    if (pivotObj.pivotValues[k] && pivotObj.pivotValues[k][0] !==
undefined) {
      rowIndx = (pivotObj.pivotValues[k][0]).rowIndex;
      break;
    }
  }
  let rowHeaders =
[0].slice.call(cTable[2].children[1].querySelectorAll('td'));
  let rows = pivotObj.dataSourceSettings.rows;
  if (rowHeaders.length > 1) {
    for (let i = 0, Cnt = rows; i < Cnt.length; i++) {
      let fields = {};
      let fieldHeaders = [];
      for (let j = 0, Lnt = rowHeaders; j < Lnt.length; j++) {
        let header = rowHeaders[j];
        if (header.className.indexOf('e-gtot') === -1 &&
header.className.indexOf('e-rowsheader') > -1 &&
header.getAttribute('fieldname') === rows[i].name) {
          var headerName =
rowHeaders[j].getAttribute('fieldname') + '_' + rowHeaders[j].textContent;
          fields[rowHeaders[j].textContent] = j;
          fieldHeaders.push(rowHeaders[j].textContent);
        }
      }
      if (i === 0) {
        for (let rnt = 0, Lnt = fieldHeaders; rnt < Lnt.length;
rnt++) {
          if (rnt !== 0) {
            let row = fields[fieldHeaders[rnt]];
            let prevRow = fields[fieldHeaders[rnt - 1]];

```

```

        for (let j = 0, ci = 1; j < cLen && ci < colLen;
j++, ci++) {
            let node =
cTable[3].children[1].children[row].childNodes[j];
            let prevNode =
cTable[3].children[1].children[prevRow].childNodes[j];
            let ri = undefined;
            let prevRi = undefined;
            if (node) {
                ri = node.getAttribute("index");
            }
            if (prevNode) {
                prevRi = prevNode.getAttribute("index");
            }
            if (ri && ri <
[].slice.call(pivotObj.pivotValues).length) {
                if
(pivotObj.pivotValues[prevRi][ci].value > pivotObj.pivotValues[ri][ci].value
&&
                    node.querySelector('.tempwrap')) {
                        let trendElement =
node.querySelector('.tempwrap');
                        trendElement.className =
trendElement.className.replace('sb-icon-neutral', 'sb-icon-loss');
                }
                else if
(pivotObj.pivotValues[prevRi][ci].value < pivotObj.pivotValues[ri][ci].value
&&
                    node.querySelector('.tempwrap')) {
                        let trendElement =
node.querySelector('.tempwrap');
                        trendElement.className =
trendElement.className.replace('sb-icon-neutral', 'sb-icon-profit');
                }
            }
        }
    }
}
}
}
}
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```
import { IDataOptions, IDataSet, Inject, PivotViewComponent } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { renewableEnergy } from '../datasource';
function App() {
  function cellTemplate(props): JSX.Element {
```

```

    return (<span className="tempwrap sb-icon-neutral pv-icons"></span>);
  }
  let dataSourceSettings: IDataOptions = {
    dataSource: renewableEnergy as IDataset[],
    expandAll: true,
    enableSorting: true,
    drilledMembers: [{ name: 'Year', items: ['FY 2015', 'FY 2017', 'FY
2018'] }],
    formatSettings: [{ name: 'ProCost', format: 'C0' }],
    rows: [
      { name: 'Year', caption: 'Production Year' }
    ],
    columns: [
      { name: 'EnerType', caption: 'Energy Type' },
      { name: 'EneSource', caption: 'Energy Source' }
    ],
    values: [
      { name: 'ProCost', caption: 'Revenue Growth' }
    ],
    filters: []
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
cellTemplate={cellTemplate.bind(this)}
dataBound={trend.bind(this)}></PivotViewComponent>);
  function trend(): void {
    let cTable: HTMLElement[] =
[[]].slice.call(document.getElementsByClassName("e-table"));
    let colLen: number = pivotObj.pivotValues[3].length;
    let cLen: number = cTable[3].children[0].children.length;
    let rLen: number = cTable[3].children[1].children.length;
    let rowIndx: number;
    for (let k: number = 0; k < rLen; k++) {
      if (pivotObj.pivotValues[k] && pivotObj.pivotValues[k][0] !==
undefined) {
        rowIndx = ((pivotObj.pivotValues[k][0]) as
IAxisSet).RowIndex;
        break;
      }
    }
    let rowHeaders: HTMLElement[] =
[[]].slice.call(cTable[2].children[1].querySelectorAll('td'));
    let rows: IFieldOptions[] = pivotObj.dataSourceSettings.rows as
IFieldOptions[];
    if (rowHeaders.length > 1) {
      for (let i: number = 0, Cnt = rows; i < Cnt.length; i++) {
        let fields: any = {};
        let fieldHeaders: any = [];
        for (let j: number = 0, Lnt = rowHeaders; j < Lnt.length;
j++) {
          let header: any = rowHeaders[j];
          if (header.className.indexOf('e-gtot') === -1 &&
header.className.indexOf('e-rowsheader') > -1 &&
header.getAttribute('fieldname') === rows[i].name) {

```

```

        var headerName =
rowHeaders[j].getAttribute('fieldname') + '_' + rowHeaders[j].textContent;
        fields[rowHeaders[j].textContent] = j;
        fieldHeaders.push(rowHeaders[j].textContent);
    }
}
    if (i === 0) {
        for (let rnt: number = 0, Lnt = fieldHeaders; rnt <
Lnt.length; rnt++) {
            if (rnt !== 0) {
                let row: number = fields[fieldHeaders[rnt]];
                let prevRow: number = fields[fieldHeaders[rnt -
1]];

                for (let j: number = 0, ci = 1; j < cLen && ci <
colLen; j++ , ci++) {
                    let node: HTMLElement =
cTable[3].children[1].children[row].childNodes[j] as HTMLElement;
                    let prevNode: HTMLElement =
cTable[3].children[1].children[prevRow].childNodes[j] as HTMLElement;
                    let ri: any = undefined;
                    let prevRi: any = undefined;
                    if (node) {
                        ri = node.getAttribute("index");
                    }
                    if (prevNode) {
                        prevRi = prevNode.getAttribute("index");
                    }
                    if (ri && ri <
[].slice.call(pivotObj.pivotValues).length) {
                        if ((pivotObj.pivotValues[prevRi][ci] as
IAxisSet).value > (pivotObj.pivotValues[ri][ci] as IAxisSet).value &&
                            node.querySelector('.tempwrap')) {
                            let trendElement: HTMLElement =
node.querySelector('.tempwrap');
                            trendElement.className =
trendElement.className.replace('sb-icon-neutral', 'sb-icon-loss');
                        } else if
((pivotObj.pivotValues[prevRi][ci] as IAxisSet).value <
(pivotObj.pivotValues[ri][ci] as IAxisSet).value &&
                            node.querySelector('.tempwrap')) {
                            let trendElement: HTMLElement =
node.querySelector('.tempwrap');
                            trendElement.className =
trendElement.className.replace('sb-icon-neutral', 'sb-icon-profit');
                        }
                    }
                }
            }
        }
    }
}
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```


Events

QueryCellInfo

The event `queryCellInfo` triggers while rendering each row and value cells in the pivot table. It allows the user to customize the current cell like adding or removing styles, editing value, etc. It has the following parameters:

- `cell` - It holds the current cell information.
- `data` - It holds the entire row data besides the current cell.
- `column` - It holds the entire column data besides the current cell.
- `pivotview` - It holds pivot table instance.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        columnWidth: 140,
        queryCellInfo: queryCellInfo.bind(this)
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    function queryCellInfo(args) {
        //triggers every time for value cell while rendering
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
```

```
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
  let gridSettings: GridSettings = {
    columnWidth: 140,
    queryCellInfo: queryCellInfo.bind(this)
  } as GridSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  function queryCellInfo(args: QueryCellInfoEventArgs): void {
    //triggers every time for value cell while rendering
  }

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

HeaderCellInfo

The event `headerCellInfo` triggers while rendering each column header cell in the pivot table. It allows the user to customize the element of the current header cell like adding or removing styles, editing value, etc. It has the following parameters:

- `node` - It holds the current header cell information

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let gridSettings = {
    columnWidth: 140,
    headerCellInfo: headerCellInfo.bind(this)
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
```

```

        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    function headerCellInfo(args) {
        //triggers every time for header cell while rendering
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
    let gridSettings: GridSettings = {
        columnWidth: 140,
        headerCellInfo: headerCellInfo.bind(this)
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    function headerCellInfo(args: HeaderCellInfoEventArgs): void {
        //triggers every time for header cell while rendering
    }

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;

```

```
ReactDOM.render(<App />, document.getElementById('sample'));
```

ColumnRender

The event [columnRender](#) triggers while framing each columns for rendering in the pivot table. It allows the user to customize the text alignment, column visibility, autofit, re-ordering, minimum and maximum width for a specific column. It has the following parameters:

- **columns** - It holds the leaf level columns (i.e., value headers) information.
- **dataSourceSettings** - It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **name** - It holds the name of the event.
- **stackedColumns** - It holds the drilled columns (i.e., including column and value headers) information.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        columnRender: columnRender.bind(this)
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    function columnRender(args) {
        for (let i = 0; i < args.columns.length; i++) {
            args.columns[i].autoFit = true;
            args.columns[i].textAlign = "Right";
        }
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
  let gridSettings: GridSettings = {
    columnRender: columnRender.bind(this)
  } as GridSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
  };
  let pivotObj: PivotViewComponent;

  function columnRender(args) {
    for (let i: number = 0; i < args.columns.length; i++) {
      args.columns[i].autoFit = true;
      args.columns[i].textAlign = "Right";
    }
  }

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} gridSettings={gridSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

CellClick

The event [cellClick](#) triggers while clicking a cell in the pivot table. For instance, using this event end-user can either add or remove styles, edit value and also perform any other DOM manipulations. It has the following parameters:

- **currentCell** - It holds the current cell information.
- **data** - It holds the clicked cell's data like axis, formatted text, actual text, row header, column header and value informations.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
```

```

    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C2' }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    };
    let pivotObj;
    function cellClick(args) {
        //triggers for every cell click in pivot table
        args.currentCell.setAttribute("style", "background-color: red;");
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} cellClick={cellClick.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent, CellClickEventArgs }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C2' }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    }
    let pivotObj: PivotViewComponent;
    function cellClick(args: CellClickEventArgs): void {
        //triggers for every cell click in pivot table
        args.currentCell.setAttribute("style", "background-color: red;")
    }
}

```

```

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} cellClick={cellClick.bind(this)}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

See Also

- [Show/hide tooltip](#)
- [Perform cell selection and get selected cells information](#)

Show hide totals in React Pivotview component

Show or hide grand totals

Allows to show or hide grand totals in rows and columns using the [showGrandTotals](#) property. To hide the grand totals in rows and columns, set the property [showGrandTotals](#) in [dataSourceSettings](#) to **false**. End user can also hide grand totals for row or columns separately by setting the property [showRowGrandTotals](#) or [showColumnGrandTotals](#) in [dataSourceSettings](#) to **false** respectively.

By default, [showGrandTotals](#), [showRowGrandTotals](#) and [showColumnGrandTotals](#) properties in [dataSourceSettings](#) are set as **true**.

INDEX.JSX

```

import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }],
        showGrandTotals: false
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}>
</PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    showGrandTotals: false
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}
><Inject services={[FieldList]}/> </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Show grand totals at top or bottom

Allows to show grand totals either at top or bottom in rows and columns using the [showGrandTotals](#) property. To show the grand totals at top in rows and columns, set the [showGrandTotals](#) property in [dataSourceSettings](#) to **Top**.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    grandTotalsPosition: 'Top',
  };
};
```



```

    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}/>
</PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        grandTotalsPosition: 'Top',
    }
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}
><Inject services={[FieldList]}/> </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Show or hide sub-totals

Allows to show or hide sub-totals in rows and columns using the [showSubTotals](#) property. To hide all the sub-totals in rows and columns, set the property [showSubTotals](#) in [dataSourceSettings](#) to **false**. End user can also hide sub-totals for rows or columns separately by setting the property [showRowSubTotals](#) or [showColumnSubTotals](#) in [dataSourceSettings](#) to **false** respectively.

By default, [showSubTotals](#), [showRowSubTotals](#) and [showColumnSubTotals](#) properties in [dataSourceSettings](#) are set as **true**.

INDEX.JSX

```

import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    allowLabelFilter: true,
    allowValueFilter: true,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    showSubTotals: false
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}/>
</PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    allowLabelFilter: true,
    allowValueFilter: true,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    showSubTotals: false
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}
><Inject services={[FieldList]}/> </PivotViewComponent>);
};
export default App;
```

```
ReactDOM.render(<App />, document.getElementById('sample'));
```

Show or hide sub-totals for specific fields

Allows to show or hide sub-totals for specific fields in rows and columns using the [showSubTotals](#) property. To hide sub-totals for a specific field in row or column axis, set the property [showSubTotals](#) in [row](#) or [column](#) to **false** respectively.

By default, [showSubTotals](#) property in [row](#) or [column](#) is set as **true**.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year', showSubTotals:
false }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country', showSubTotals: false }, { name: 'Products'
}],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}/>
</PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year', showSubTotals:
false }, { name: 'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}/>
</PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

```

    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country', showSubTotals: false }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}
><Inject services={[FieldList]}/> </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Show or hide totals using toolbar

It can also be achieved using built-in toolbar options by setting the [showToolbar](#) property to **true**. Also, include the items **GrandTotal** and **SubTotal** within the [toolbar](#) property. End user can now see "Show/Hide Grand totals" and "Show/Hide Sub totals" icons in toolbar UI automatically.

INDEX.JSX

```

{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent, Toolbar, Inject } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  let toolbarOptions = ['SubTotal', 'GrandTotal'];
  return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
gridSettings={{ columnWidth: 140 }} showToolbar={true} displayOption={{
view: 'Both' }} toolbar={toolbarOptions}><Inject
services={[Toolbar]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  PivotViewComponent, IDataOptions, Toolbar, Inject
} from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj: PivotViewComponent;
  let toolbarOptions: any = ['SubTotal', 'GrandTotal'];

  return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
gridSettings={{ columnWidth: 140 }} showToolbar={true} displayOption={{
view: 'Both' }} toolbar={toolbarOptions}><Inject services={[Toolbar]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}
```

Hyper link in React Pivotview component

The pivot table supports to show hyperlink option to link data for individual cells that are displayed in the component. Also, the hyperlink can be enabled separately for row headers, column headers, value cells, and summary cells using the [hyperlinkSettings](#). It can be configured through code behind, during initial rendering and the settings available to show hyperlink are:

- [showHyperlink](#): It allows to set the visibility of hyperlink in all cells.
- [showRowHeaderHyperlink](#): It allows to set the visibility of hyperlink in row headers.
- [showColumnHeaderHyperlink](#): It allows to set the visibility of hyperlink in column headers.
- [showValueCellHyperlink](#): It allows to set the visibility of hyperlink in value cells.
- [showSummaryCellHyperlink](#): It allows to set the visibility of hyperlink in summary cells.
- [headerText](#): It allows to set the visibility of hyperlink based on header text.
- [conditionalSettings](#): It allows to set the visibility of hyperlink based on specific condition.

By default, the hyperlink options are disabled for all cells in the pivot table.

User defined style can be applied to hyperlink using [cssClass](#) property in [hyperlinkSettings](#).

Hyperlink for all cells

The pivot table has an option to show hyperlink option for all cells that are currently in display. To do so, user need to set [showHyperlink](#) to **true**.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        dataSource: pivotData,
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
    showHyperlink: true,
    cssClass: 'e-custom-class'
}}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataset[],
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    }
}
```

```

let pivotObj: PivotViewComponent;
return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
  showHyperlink: true,
  cssClass: 'e-custom-class'
}}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Hyperlink for row headers

The pivot table has an option to show hyperlink option for row header cells alone that are currently in display. To do so, user need to set [showRowHeaderHyperlink](#) to **true**.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    dataSource: pivotData,
    expandAll: false,
    enableSorting: true,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  };
  let pivotObj;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
  showRowHeaderHyperlink: true,
  cssClass: 'e-custom-class'
}}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],

```

```

    expandAll: false,
    enableSorting: true,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
  showRowHeaderHyperlink: true,
  cssClass: 'e-custom-class'
}}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Hyperlink for column headers

The pivot table has an option to show hyperlink option for column header cells alone that are currently in display. To do so, user need to set [showColumnHeaderHyperlink](#) to **true**.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    dataSource: pivotData,
    expandAll: false,
    enableSorting: true,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  };
  let pivotObj;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
  showColumnHeaderHyperlink: true,
  cssClass: 'e-custom-class'
}}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```


INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: false,
    enableSorting: true,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
  showColumnHeaderHyperlink: true,
  cssClass: 'e-custom-class'
}}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Hyperlink for value cells

The pivot table has an option to show hyperlink option for value cells alone that are currently in display. To do so, user need to set [showValueCellHyperlink](#) to **true**.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    dataSource: pivotData,
    expandAll: false,
    enableSorting: true,
    drilledMembers: [{ name: 'Year', items: ['FY 2015' ]}, { name:
'Country', items: ['France' ]}],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
```

```

        filters: []
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
    showValueCellHyperlink: true,
    cssClass: 'e-custom-class'
}}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataset[],
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Year', items: ['FY 2015'] }, { name:
'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    }
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
    showValueCellHyperlink: true,
    cssClass: 'e-custom-class'
}}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Hyperlink for summary cells

The pivot table has an option to show hyperlink option for summary cells alone that are currently in display. To do so, user need to set [showSummaryCellHyperlink](#) to **true**.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```

import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    dataSource: pivotData,
    expandAll: false,
    enableSorting: true,
    drilledMembers: [{ name: 'Year', items: ['FY 2015'] }, { name:
'Country', items: ['France'] }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  };
  let pivotObj;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
  showSummaryCellHyperlink: true,
  cssClass: 'e-custom-class'
}}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: false,
    enableSorting: true,
    drilledMembers: [{ name: 'Year', items: ['FY 2015'] }, { name:
'Country', items: ['France'] }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  }
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
  showSummaryCellHyperlink: true,
  cssClass: 'e-custom-class'
}}></PivotViewComponent>);
};

```

```
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Condition based hyperlink

The pivot table has an option to show hyperlink option to the cells based on specific conditions. It can be configured using the [conditionalSettings](#) option through code behind, during initial rendering. The settings required to sort are:

- [measure](#): Specifies the value field name to get visibility of hyperlink option for specific measure.
- [conditions](#): Specifies the operator type such as equals, greater than, less than, etc.
- [value1](#): Specifies the start value.
- [value2](#): Specifies the end value.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        dataSource: pivotData,
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Year', items: ['FY 2015'] }, { name:
'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    let hyperlinkSettings = {
        conditionalSettings: [{
            measure: 'Sold',
            conditions: 'Between',
            value1: 150,
            value2: 500
        }],
        cssClass: 'e-custom-class'
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings}
hyperlinkSettings={hyperlinkSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { HyperLinkSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/hyperlinksettings';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: false,
    enableSorting: true,
    drilledMembers: [{ name: 'Year', items: ['FY 2015'] }, { name:
'Country', items: ['France'] }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  }
  let hyperlinkSettings: HyperLinkSettings = {
    conditionalSettings: [{
      measure: 'Sold',
      conditions: 'Between',
      value1: 150,
      value2: 500
    }],
    cssClass: 'e-custom-class'
  } as HyperLinkSettings;
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings}
hyperlinkSettings={hyperlinkSettings}></PivotViewComponent>
);
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Header based hyperlink

The pivot table has an option to show hyperlink in the cells based on specific row or column header. It can be configured using the [headerText](#) option through code behind, during initial rendering.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    dataSource: pivotData,
    expandAll: false,
    enableSorting: true,

```

```

        drilledMembers: [{ name: 'Year', items: ['FY 2015'] }, { name:
'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    let pivotObj;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
    headerText: 'FY 2015.Q1.Units Sold',
    cssClass: 'e-custom-class'
}}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataset[],
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Year', items: ['FY 2015'] }, { name:
'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    }
    let pivotObj: PivotViewComponent;
    return (<PivotViewComponent height={350} ref={d => pivotObj = d}
id='PivotView' dataSourceSettings={dataSourceSettings} hyperlinkSettings={{
    headerText: 'FY 2015.Q1.Units Sold',
    cssClass: 'e-custom-class'
}}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Event

The event [hyperlinkCellClick](#) fires on every hyperlink cell click.

It has following parameters - **Cancel** and **CurrentCell**. The parameter **CurrentCell** is used to customize the host cell element by any means. Meanwhile, when the parameter **Cancel** is set to **true**, applied customization will not be updated to the host cell element.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        allowLabelFilter: true,
        allowValueFilter: true,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' } ]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
    height={350} dataSourceSettings={dataSourceSettings}
    hyperlinkCellClick={hyperlinkCellClick.bind(this)} showFieldList={true}
    hyperlinkSettings={{
        showHyperlink: true,
        cssClass: 'e-custom-class'
    }}>
        <Inject services={[FieldList]} />
    </PivotViewComponent>);
    function hyperlinkCellClick(args) {
        args.cancel = false;
        args.currentCell.setAttribute("data-url",
        "https://ej2.syncfusion.com/"); //here we have redirected to EJ2 Syncfusion
        on hyperlinkcell click
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent,
HyperCellClickEventArgs } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
```

```

    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    allowLabelFilter: true,
    allowValueFilter: true,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France'] }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
hyperlinkCellClick={hyperlinkCellClick.bind(this)} showFieldList={true}
hyperlinkSettings={{
  showHyperlink: true,
  cssClass: 'e-custom-class'
}}>
    <Inject services={[FieldList]} />
  </PivotViewComponent>);
  function hyperlinkCellClick(args: HyperCellClickEventArgs): void {
    args.cancel = false;
    args.currentCell.setAttribute("data-url",
"https://ej2.syncfusion.com/");//here we have redirected to EJ2 Syncfusion
on hyperlinkcell click
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

See Also

- [Apply condition based hyperlink for specific row or column](#)

Tool bar in React Pivotview component

A toolbar option has been provided to the pivot controls. It allows users access the frequently used features easily. Users also can save the state of the pivot table and load it back whenever required using this option. This option can be enabled by setting the `showToolbar` property to true. The [toolbar](#) property accepts the collection of built-in toolbar options.

To use the toolbar, inject the `Toolbar` module into the pivot table.

Built-in toolbar options

Built-in toolbar options can be added by defining the [toolbar](#) as a collection of built-in options.

The following table shows built-in toolbar options and its actions.

Built-in Toolbar Options	Actions
----- -----	
New	Creates a new report

- | Save | Saves the current report |
- | Save As | Save as current report |
- | Rename | Renames the current report |
- | Delete | Deletes the current report |
- | Load | Loads any report from the report list |
- | Grid | Shows pivot table |
- | Chart | Shows a chart in any type from the built-in list and option to enable/disable multiple axes|
- | Exporting | Exports the pivot table as PDF/Excel/CSV |
- | Sub total | Shows or hides sub totals |
- | Grand total | Shows or hides grand totals |
- | Conditional Formatting | Shows the conditional formatting pop-up to apply formatting |
- | Number Formatting | Shows the number formatting pop-up to apply number formatting |
- | Fieldlist | Shows the fieldlist pop-up |
- | MDX | Shows the MDX query that was run to retrieve data from the OLAP data source. **NOTE: This applies only to the OLAP data source.** |

Report manipulation like save, load, rename, etc., operations can be performed through events. In the following example, the localStorage (session storage) is used to manipulate the report operation.

INDEX.JSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent, Inject, FieldList, CalculatedField, Toolbar,
PDFExport, ExcelExport, ConditionalFormatting, NumberFormatting } from
'@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  let toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
  'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'];
  function saveReport(args) {
    let reports = [];
```

```

        let isSaved = false;
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= "" ) {
            reports = JSON.parse(localStorage.pivotviewReports);
        }
        if (args.report && args.reportName && args.reportName !== '') {
            reports.map(function (item) {
                if (args.reportName === item.reportName) {
                    item.report = args.report;
                    isSaved = true;
                }
            });
            if (!isSaved) {
                reports.push(args);
            }
            localStorage.pivotviewReports = JSON.stringify(reports);
        }
    }
    function fetchReport(args) {
        let reportCollection = [];
        let reeportList = [];
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= "") {
            reportCollection = JSON.parse(localStorage.pivotviewReports);
        }
        reportCollection.map(function (item) {
            reeportList.push(item.reportName);
        });
        args.reportName = reeportList;
    }
    function loadReport(args) {
        let reportCollection = [];
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= "") {
            reportCollection = JSON.parse(localStorage.pivotviewReports);
        }
        reportCollection.map(function (item) {
            if (args.reportName === item.reportName) {
                args.report = item.report;
            }
        });
        if (args.report) {
            pivotObj.dataSource = JSON.parse(args.report).dataSource;
        }
    }
    function removeReport(args) {
        let reportCollection = [];
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= "") {
            reportCollection = JSON.parse(localStorage.pivotviewReports);
        }
        for (let i = 0; i < reportCollection.length; i++) {
            if (reportCollection[i].reportName === args.reportName) {
                reportCollection.splice(i, 1);
            }
        }
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= "") {

```

```

        localStorage.pivotviewReports =
JSON.stringify(reportCollection);
    }
}
function renameReport(args) {
    let reportCollection = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= " ") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item) { if (args.reportName ===
item.reportName) {
        item.reportName = args.rename;
    } });
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= " ") {
        localStorage.pivotviewReports =
JSON.stringify(reportCollection);
    }
}
function newReport() {
    pivotObj.setProperties({ dataSource: { columns: [], rows: [],
values: [], filters: [] } }, false);
}
return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
showFieldList={true} gridSettings={{ columnWidth: 140 }}
allowExcelExport={true} allowConditionalFormatting={true}
allowNumberFormatting={true} allowPdfExport={true} showToolbar={true}
allowCalculatedField={true} displayOption={{ view: 'Both' }}
toolbar={toolbarOptions} newReport={newReport.bind(this)}
renameReport={renameReport.bind(this)}
removeReport={removeReport.bind(this)} loadReport={loadReport.bind(this)}
fetchReport={fetchReport.bind(this)}
saveReport={saveReport.bind(this)}><Inject services={[FieldList,
CalculatedField, Toolbar, PDFExport, ExcelExport, ConditionalFormatting,
NumberFormatting]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    PivotViewComponent, IDataOptions, Inject, FieldList, CalculatedField,
    Toolbar, PDFExport, ExcelExport, ConditionalFormatting, SaveReportArgs,
    FetchReportArgs, LoadReportArgs, RemoveReportArgs, RenameReportArgs,
    NumberFormatting
} from '@syncfusion/ej2-react-pivotview';
import { pivotData } from '../datasource';
function App() {

```

```

let dataSourceSettings: IDataOptions = {
  columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
  dataSource: pivotData as IDataset[],
  expandAll: false,
  filters: [],
  drilledMembers: [{ name: 'Country', items: ['France' ]}],
  formatSettings: [{ name: 'Amount', format: 'C0' }],
  rows: [{ name: 'Country' }, { name: 'Products' }],
  values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
};
let pivotObj: PivotViewComponent;
let toolbarOptions: any = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
  'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'];
function saveReport(args: any): void {
  let reports: SaveReportArgs[] = [];
  let isSaved: boolean = false;
  if (localStorage.pivotviewReports && localStorage.pivotviewReports
!== "") {
    reports = JSON.parse(localStorage.pivotviewReports);
  }
  if (args.report && args.reportName && args.reportName !== '') {
    reports.map(function (item: any): any {
      if (args.reportName === item.reportName) {
        item.report = args.report; isSaved = true;
      }
    });
    if (!isSaved) {
      reports.push(args);
    }
    localStorage.pivotviewReports = JSON.stringify(reports);
  }
}
function fetchReport(args: FetchReportArgs): void {
  let reportCollection: string[] = [];
  let reeportList: string[] = [];
  if (localStorage.pivotviewReports && localStorage.pivotviewReports
!== "") {
    reportCollection = JSON.parse(localStorage.pivotviewReports);
  }
  reportCollection.map(function (item: any): void {
    reeportList.push(item.reportName); });
  args.reportName = reeportList;
}
function loadReport(args: LoadReportArgs): void {
  let reportCollection: string[] = [];
  if (localStorage.pivotviewReports && localStorage.pivotviewReports
!== "") {
    reportCollection = JSON.parse(localStorage.pivotviewReports);
  }
  reportCollection.map(function (item: any): void {
    if (args.reportName === item.reportName) {
      args.report = item.report;
    }
  })
}

```

```

    });
    if (args.report) {
        pivotObj.dataSource = JSON.parse(args.report).dataSource;
    }
}
function removeReport(args: RemoveReportArgs): void {
    let reportCollection: any[] = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
    !== "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    for (let i: number = 0; i < reportCollection.length; i++) {
        if (reportCollection[i].reportName === args.reportName) {
            reportCollection.splice(i, 1);
        }
    }
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
    !== "") {
        localStorage.pivotviewReports =
        JSON.stringify(reportCollection);
    }
}
function renameReport(args: RenameReportArgs): void {
    let reportCollection: string[] = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
    !== "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item: any): any { if (args.reportName
    === item.reportName) { item.reportName = args.rename; } });
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
    !== "") {
        localStorage.pivotviewReports =
        JSON.stringify(reportCollection);
    }
}
function newReport(): void {
    pivotObj.setProperties({ dataSource: { columns: [], rows: [],
    values: [], filters: [] } }, false);
}

return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
showFieldList={true} gridSettings={{ columnWidth: 140 }}
allowExcelExport={true} allowConditionalFormatting={true}
allowNumberFormatting={true} allowPdfExport={true} showToolBar={true}
allowCalculatedField={true} displayOption={{ view: 'Both' }}
toolbar={toolbarOptions} newReport={newReport.bind(this)}
renameReport={renameReport.bind(this)}
removeReport={removeReport.bind(this)} loadReport={loadReport.bind(this)}
fetchReport={fetchReport.bind(this)}
saveReport={saveReport.bind(this)}><Inject services={[FieldList,
CalculatedField, Toolbar, PDFExport, ExcelExport, ConditionalFormatting,
NumberFormatting]} /></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

```
{% endraw %}
```

Show desired chart types in the dropdown menu

By default, all chart types are displayed in the dropdown menu included in the toolbar. However, based on the request for an application, we may need to show selective chart types on our own. This can be achieved using the [chartTypes](#) property. To know more about supporting chart types, [click here](#).

INDEX.JSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent, Inject, Toolbar } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: pivotData,
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' } ]
  };
  let pivotObj;
  let toolbarOptions = ['Grid', 'Chart'];
  let chartTypes = ['Column', 'Bar', 'Line', 'Area'];
  return (<PivotViewComponent id='PivotView'
    dataSourceSettings={dataSourceSettings} width={100%} height={350}
    showToolbar={true} toolbar={toolbarOptions} displayOption={{ view: 'Both' }}
    chartTypes={chartTypes}><Inject
    services={[Toolbar]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  PivotViewComponent, IDataOptions, Inject, Toolbar
} from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: pivotData as IDataset[],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
  };
  let pivotObj;
  let toolbarOptions = ['Grid', 'Chart'];
  let chartTypes = ['Column', 'Bar', 'Line', 'Area'];
  return (<PivotViewComponent id='PivotView'
    dataSourceSettings={dataSourceSettings} width={100%} height={350}
    showToolbar={true} toolbar={toolbarOptions} displayOption={{ view: 'Both' }}
    chartTypes={chartTypes}><Inject
    services={[Toolbar]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}
```

```

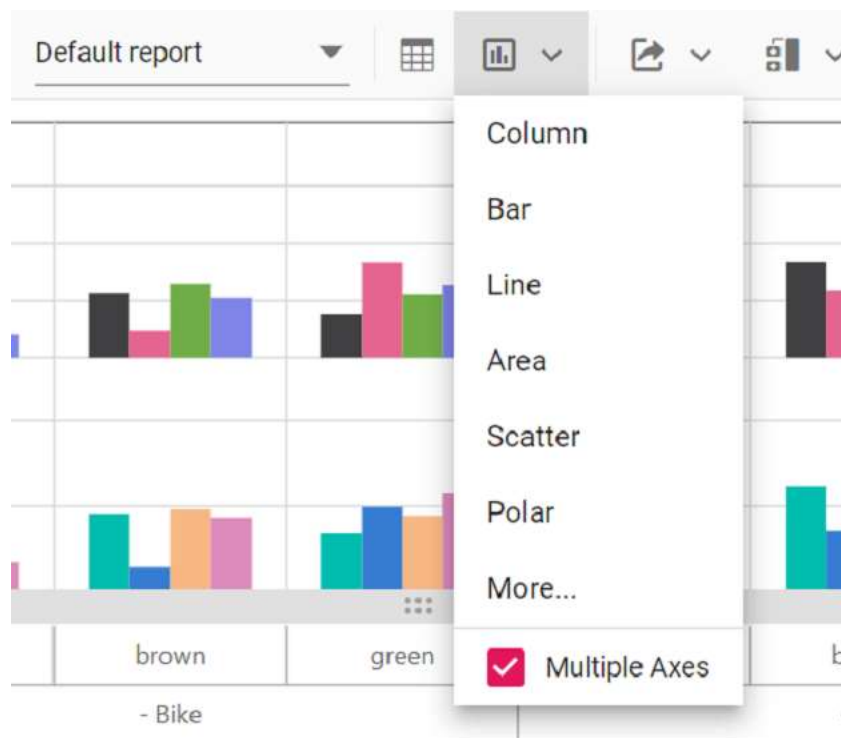
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj: PivotViewComponent;
  let toolbarOptions: any = ['Grid', 'Chart'];
  let chartTypes: any = ['Column', 'Bar', 'Line', 'Area'];

  return (<PivotViewComponent id='PivotView'
dataSourceSettings={dataSourceSettings} width={100%} height={350}
showToolbar={true} toolbar={toolbarOptions} displayOption={{ view: 'Both' }}
chartTypes={chartTypes} ><Inject services={[ Toolbar]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

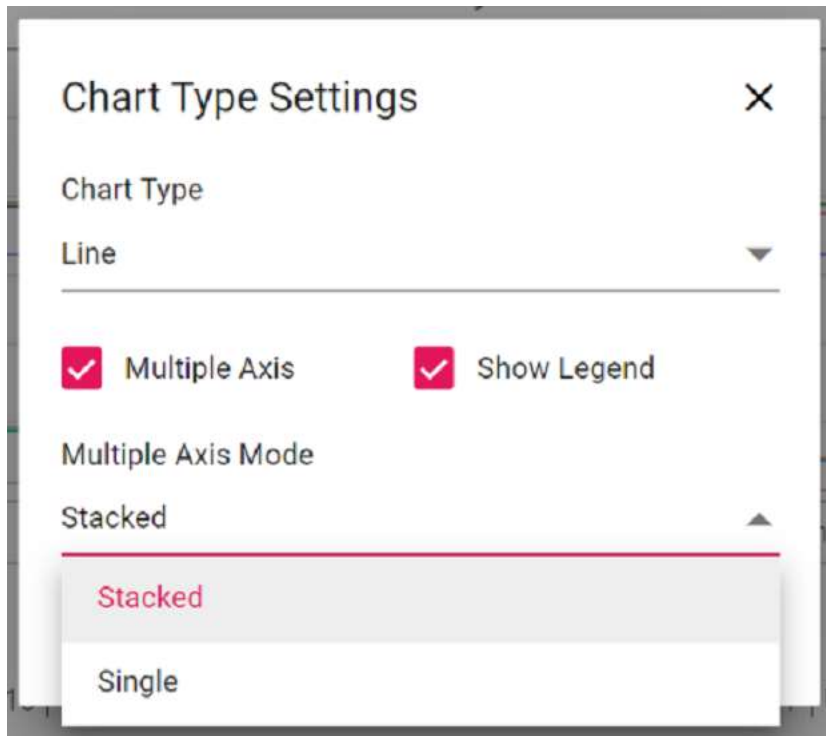
Switch the chart to multiple axes

In the chart, the user can switch from single axis to multiple axes with the help of the built-in checkbox available inside the chart type dropdown menu in the toolbar. For more information [refer here](#).



<!-- markdownlint-disable MD009 -->

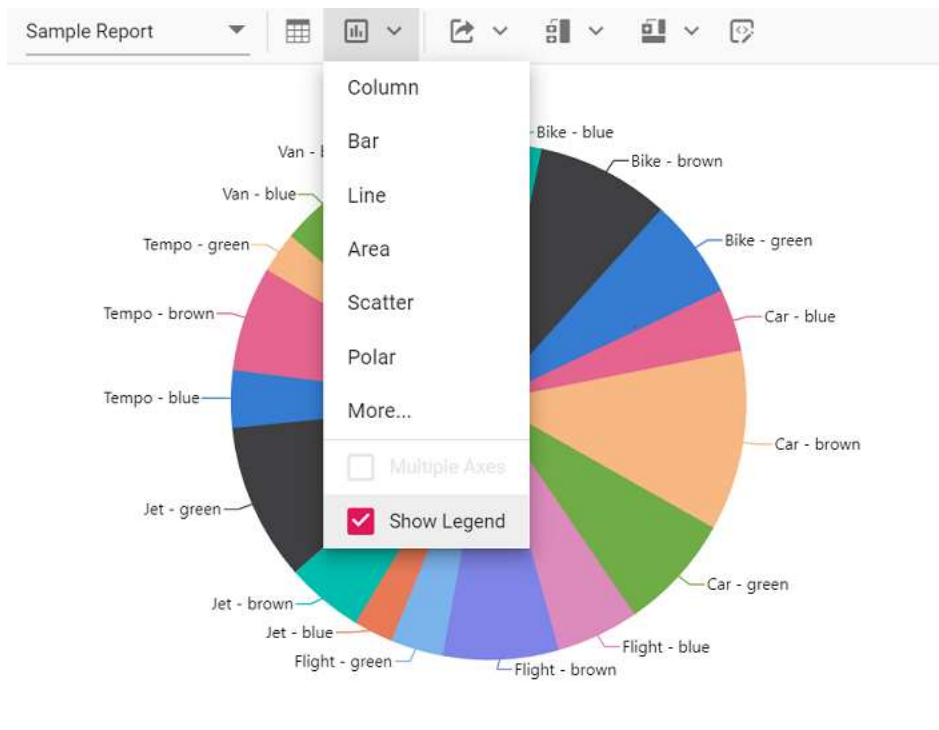
There are two modes available in **Multiple Axis** option: **Stacked** and **Single**. The modes can be changed using “Multiple Axis Mode” drop-down list which appears while clicking the **More...** option.



Show or hide legend

In the chart, legend can be shown or hidden dynamically with the help of the built-in option available in the chart type drop-down menu.

By default, the legend is not be visible for the accumulation chart types like pie, doughnut, pyramid, and funnel. Users can enable or disable using the built-in checkbox option.



Adding custom option to the toolbar

In addition to the existing built-in toolbar items, new toolbar item(s) may also be included. This can be achieved by using the [toolbarRender](#) event. The action of the new toolbar item(s) can also be defined within this event.

The new toolbar item(s) can be added to the desired position in the toolbar using the `splice` option.

INDEX.JSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent, Inject, Toolbar } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' } ]
    };
    let pivotObj;
    let toolbarOptions = ['Expand/Collapse'];
    function beforeToolbarRender(args) {
        args.customToolbar.splice(12, 0, {
            prefixIcon: 'e-tool-expand e-icons', tooltipText: 'Expand/Collapse',
            click: toolbarClicked.bind(this),
        });
    }
    function toolbarClicked(args) {
        pivotObj.dataSourceSettings.expandAll = !pivotObj.dataSourceSettings.expandAll;
    }
    return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
        dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
        gridSettings={{ columnWidth: 140 }} showToolbar={true} displayOption={{
            view: 'Both' }} toolbar={toolbarOptions}
        toolbarRender={beforeToolbarRender.bind(this)}><Inject
            services={[Toolbar]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
```

```

import {
  PivotViewComponent, IDataOptions, Inject, Toolbar, ToolbarArgs
} from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj: PivotViewComponent;
  let toolbarOptions: any = ['Expand/Collapse'];
  function beforeToolbarRender(args: ToolbarArgs): void {
    args.customToolbar.splice(12, 0, {
      prefixIcon: 'e-tool-expand e-icons', tooltipText:
'Expand/Collapse',
      click: toolbarClicked.bind(this),
    });
  }
  function toolbarClicked(args: any): void {
    pivotObj.dataSourceSettings.expandAll =
!pivotObj.dataSourceSettings.expandAll;
  }

  return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
gridSettings={{ columnWidth: 140 }} showToolbar={true} displayOption={{
view: 'Both' }} toolbar={toolbarOptions}
toolbarRender={beforeToolbarRender.bind(this)}><Inject services={[Toolbar]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

In the above topic, we have seen how to add an icon as one of the toolbar item in toolbar panel. In the next topic, we are going to see how to frame the entire toolbar panel and how to add a custom control in it.

Toolbar Template

It allows to customize the toolbar panel by using template option. It allows any custom control to be used as one of the toolbar item inside the toolbar panel. It can be achieved by two ways,

Here, the entire toolbar panel can be framed in HTML elements that are appended at the top of the pivot table. The **id** of the HTML element needs to be set in the [toolbarTemplate](#) property in-order to map it to the pivot table.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```

import { PivotViewComponent, Inject, Toolbar } from '@syncfusion/ej2-react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    };
    let pivotObj;
    let toolbarOptions = '#template';
    return (<div><div><PivotViewComponent id='PivotView' ref={d => pivotObj = d} dataSourceSettings={dataSourceSettings} width={'100%'} height={350} showToolbar={true} toolbarTemplate={toolbarOptions}><Inject services={[Toolbar]}></PivotViewComponent></div><div id='template'><div><ButtonComponent id='expandall' cssClass="e-flat e-primary" onClick={expandAll.bind(this)}>EXPAND ALL</ButtonComponent><ButtonComponent id='collapseall' cssClass="e-flat e-primary" onClick={collapseAll.bind(this)}>COLLAPSE ALL</ButtonComponent></div></div></div>);
    function expandAll() {
        pivotObj.dataSourceSettings.expandAll = true;
    }
    function collapseAll() {
        pivotObj.dataSourceSettings.expandAll = false;
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    PivotViewComponent, IDataOptions, Inject, Toolbar
} from '@syncfusion/ej2-react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData as IDataset[],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    };
    let pivotObj: PivotViewComponent;

```

```

let toolbarOptions: any = '#template';

return (<div><div><PivotViewComponent id='PivotView' ref={d => pivotObj
= d} dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
showToolbar={true} toolbarTemplate={toolbarOptions} ><Inject services={[
Toolbar]} /></PivotViewComponent></div><div
id='template'><div><ButtonComponent id='expandall' cssClass="e-flat e-
primary" onClick={expandAll.bind(this)}>EXPAND
ALL</ButtonComponent><ButtonComponent id='collapseall' cssClass="e-flat e-
primary" onClick={collapseAll.bind(this)}>COLLAPSE
ALL</ButtonComponent></div></div></div>);

function expandAll() {
    pivotObj.dataSourceSettings.expandAll=true;
}
function collapseAll() {
    pivotObj.dataSourceSettings.expandAll=false;
}
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Another option allows to frame a custom toolbar item using HTML elements and include in the toolbar panel at the desired position. The custom toolbar items can be declared as control **instance** or element **id** in the [toolbar](#) property in pivot table.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent, Inject, Toolbar } from '@syncfusion/ej2-react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    };
    let pivotObj;
    let toolbarOptions = [{ template: '#enablertl' }, { template:
'#disablertl' }];
    return (<div><div><PivotViewComponent id='PivotView' ref={d => pivotObj
= d} dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
showToolbar={true} toolbar={toolbarOptions}><Inject
services={[Toolbar]} /></PivotViewComponent></div><div><ButtonComponent
id='enablertl' cssClass="e-flat e-primary"
onClick={enableRtl.bind(this)}>ENABLE
RTL</ButtonComponent></div><div><ButtonComponent id='disablertl'
cssClass="e-flat e-primary" onClick={disableRtl.bind(this)}>DISABLE
RTL</ButtonComponent></div></div>);
    function enableRtl() {

```

```

        pivotObj.enableRtl = true;
    }
    function disableRtl() {
        pivotObj.enableRtl = false;
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    PivotViewComponent, IDataOptions, Inject, Toolbar
} from '@syncfusion/ej2-react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    };
    let pivotObj: PivotViewComponent;
    let toolbarOptions: any = [{template: '#enablertl'},
{template: '#disablertl'}];

    return (<div><div><PivotViewComponent id='PivotView' ref={d => pivotObj
= d} dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
showToolbar={true} toolbar={toolbarOptions} ><Inject services={[ Toolbar]}
/></PivotViewComponent></div><div><ButtonComponent id='enablertl'
cssClass="e-flat e-primary" onClick={enableRtl.bind(this)}>ENABLE
RTL</ButtonComponent></div><div><ButtonComponent id='disablertl'
cssClass="e-flat e-primary" onClick={disableRtl.bind(this)}>DISABLE
RTL</ButtonComponent></div></div>);
    function enableRtl() {
        pivotObj.enableRtl=true;
    }
    function disableRtl() {
        pivotObj.enableRtl=false;
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Note: For both options, the actions for the toolbar template items can be defined in the event [toolbarClick](#). Also, if the toolbar item is a custom control then its built-in events can also be accessed.

<!-- markdownlint-disable MD009 -->

Save and load report as a JSON file

The current pivot report can be saved as a JSON file in the desired path and loaded back to the pivot table at any time.

INDEX.JSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
const SAMPLE_CSS = `
.fileUpload {
    position: relative;
    overflow: hidden;
    margin: 10px;
}
.fileUpload input.upload {
    position: absolute;
    top: 0;
    right: 0;
    margin: 0;
    padding: 0;
    font-size: 20px;
    cursor: pointer;
    opacity: 0;
    filter: alpha(opacity=0);
}
`;
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    function ondataBound(args) {
        var dataSource =
JSON.parse(pivotObj.getPersistData()).dataSourceSettings.dataSource;
        var a = document.getElementById('save');
        var mime_type = 'application/octet-stream'; // text/html, image/png,
et c
        a.setAttribute('download', 'pivot.JSON');
        a.href = 'data:' + mime_type + ';base64,' +
btoa(JSON.stringify(dataSource) || '');
        document.getElementById('files').addEventListener('change',
readBlob, false);
    }
    function readBlob(args) {
        var files = document.getElementById('load').files;
        var file = files[0];
        var start = 0;
        var stop = file.size - 1;
    }
}
```

```

    var reader = new FileReader();
    reader.onloadend = function (evt) {
        if (evt.target.readyState === FileReader.DONE) {
            pivotObj.dataSource = JSON.parse(evt.target.result);
        }
    };
    var blob = file.slice(start, stop + 1);
    reader.readAsBinaryString(blob);
}
return (<div className='control-pane'>
    <div><style>{SAMPLE_CSS}</style><PivotViewComponent
id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
gridSettings={{ columnWidth: 140 }}
dataBound={ondataBound.bind(this)}></PivotViewComponent></div><a id="save"
class="btn btn-primary">Save</a><div class="fileUpload btn btn-
primary"><span>Load</span><input id="files" type="file"
class="upload"/></div>
    </div>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { IDataOptions, IDataset, Inject, PivotViewComponent } from
'@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
const SAMPLE_CSS = `
.fileUpload {
    position: relative;
    overflow: hidden;
    margin: 10px;
}
.fileUpload input.upload {
    position: absolute;
    top: 0;
    right: 0;
    margin: 0;
    padding: 0;
    font-size: 20px;
    cursor: pointer;
    opacity: 0;
    filter: alpha(opacity=0);
}
`;
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        filters: [],
    };

```

```

    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj: PivotViewComponent;
  function ondataBound(args: any): void {
    var dataSource =
JSON.parse(pivotObj.getPersistData()).dataSourceSettings.dataSource;
    var a = document.getElementById('save');
    var mime_type = 'application/octet-stream'; // text/html, image/png,
et c
    a.setAttribute('download', 'pivot.JSON');
    a.href = 'data:' + mime_type + ';base64,' +
btoa(JSON.stringify(dataSource) || '');
    document.getElementById('files').addEventListener('change',
readBlob, false);
  }
  function readBlob(args: any): void {
    var files = document.getElementById('load').files;
    var file = files[0];
    var start = 0;
    var stop = file.size - 1;
    var reader = new FileReader();
    reader.onloadend = function(evt) {
      if (evt.target.readyState === FileReader.DONE) {
        pivotObj.dataSource = JSON.parse(evt.target.result);
      }
    };
    var blob = file.slice(start, stop + 1);
    reader.readAsBinaryString(blob);
  }
  return (<div className='control-pane'>
    <div><style>{SAMPLE_CSS}</style><PivotViewComponent
id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
gridSettings={{ columnWidth: 140 }}
dataBound={ondataBound.bind(this)}></PivotViewComponent></div><a id="save"
class="btn btn-primary">Save</a><div class="fileUpload btn btn-
primary"><span>Load</span><input id="files" type="file" class="upload"
/></div>
    </div>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

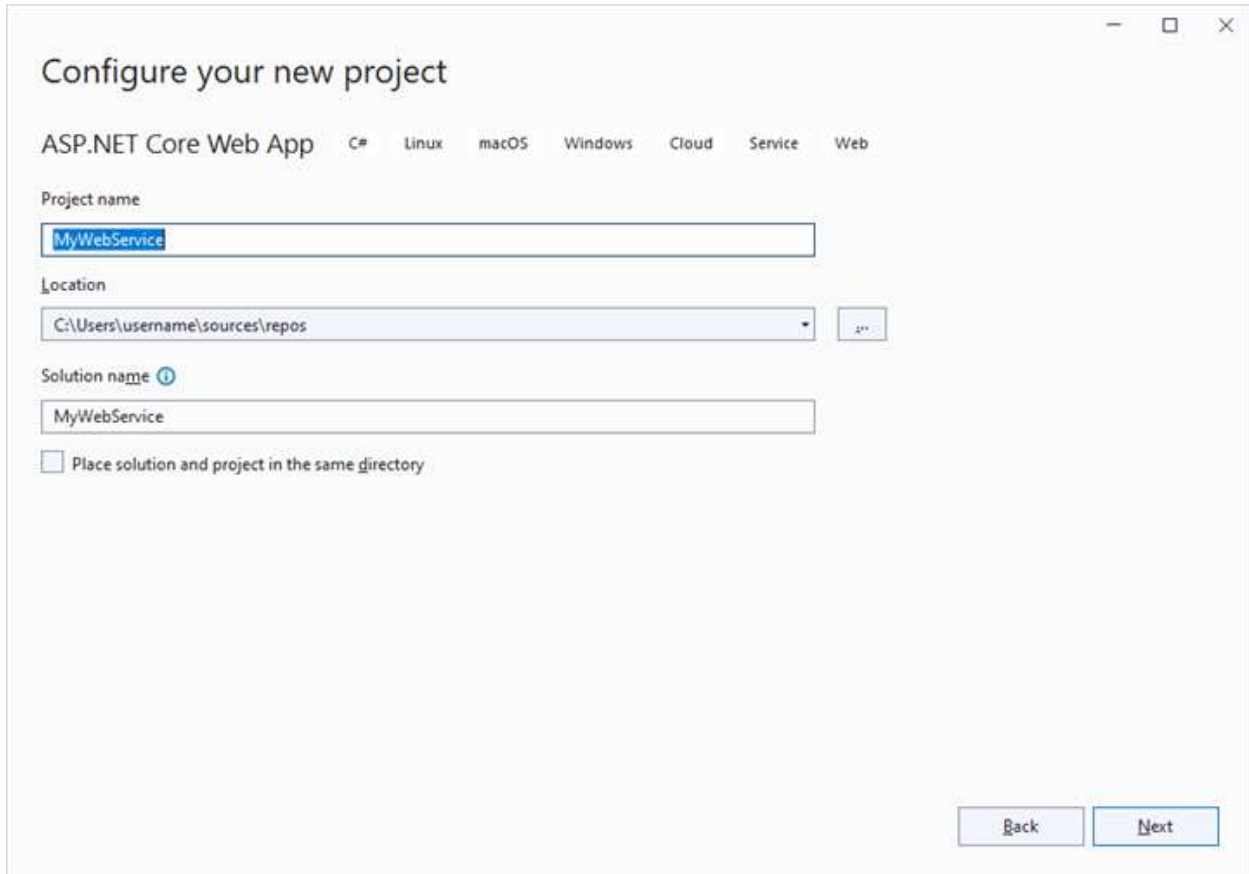
```

Save and load reports to a SQL database

SQL Server is a relational database management system (RDBMS) that can be used to store and manage large amounts of data. In this topic, we will see how to save, save as, rename, load, delete, and add reports between a SQL Server database and a React Pivot Table at runtime.

Create a Web API service to connect to a SQL Server database

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).



Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name

MyWebService

Location

C:\Users\username\source\repos

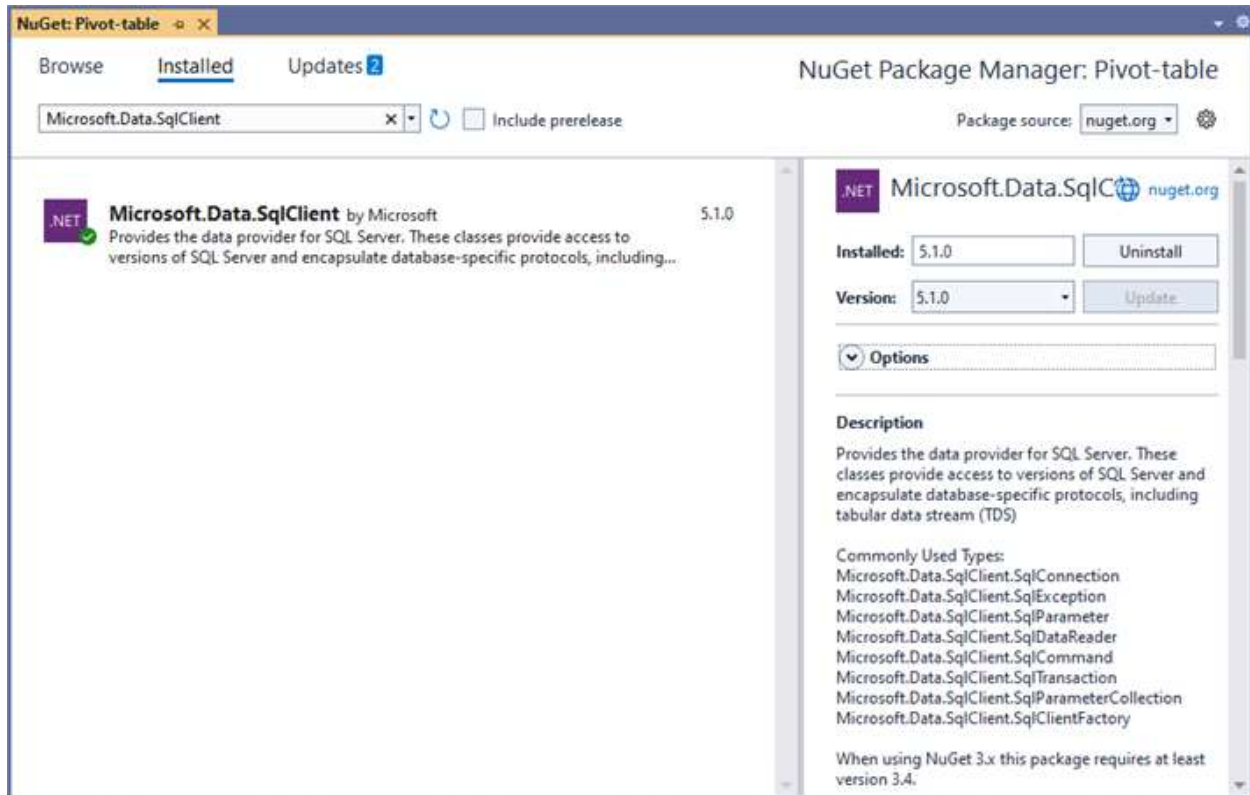
Solution name ⓘ

MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a SQL Server database using the Microsoft SqlClient in our application, we need to install the [Microsoft.Data.SqlClient](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **Microsoft.Data.SqlClient** and install it.



3. Under the **Controllers** folder, create a Web API controller (aka, PivotController.cs) file that aids in data communication with the Pivot Table.

4. In the Web API Controller (aka, PivotController), the **OpenConnection** method is used to connect to the SQL database. The **GetDataTable** method then processes the specified SQL query string, retrieves data from the database, and converts it into a **DataTable** using **SqlCommand** and **SqlDataAdapter**. This **DataTable** can be used to retrieve saved reports and modify them further as shown in the code block below.

[PivotController.cs]

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpPost]
```

```
[Route("Pivot/SaveReport")]
public void SaveReport([FromBody] Dictionary<string, string> reportArgs)
{
    SaveReportToDB(reportArgs["reportName"], reportArgs["report"]);
}
[HttpPost]
[Route("Pivot/FetchReport")]
public IActionResult FetchReport()
{
    return Ok((FetchReportListFromDB()));
}
[HttpPost]
[Route("Pivot/LoadReport")]
public IActionResult LoadReport([FromBody] Dictionary<string, string> reportArgs)
{
    return Ok((LoadReportFromDB(reportArgs["reportName"])));
}
[HttpPost]
[Route("Pivot/RemoveReport")]
public void RemoveReport([FromBody] Dictionary<string, string> reportArgs)
{
    RemoveReportFromDB(reportArgs["reportName"]);
}
[HttpPost]
[Route("Pivot/RenameReport")]
public void RenameReport([FromBody] RenameReportDB reportArgs)
{
    RenameReportInDB(reportArgs.ReportName, reportArgs.RenameReport, reportArgs.isReportExists);
}
public class RenameReportDB
{
    public string ReportName { get; set; }
    public string RenameReport { get; set; }
```

```
public bool isReportExists { get; set; }
}

private void SaveReportToDB(string reportName, string report)
{
    SqlConnection sqlConn = OpenConnection();
    bool isDuplicate = true;
    SqlCommand cmd1 = null;
    foreach (DataRow row in GetDataTable(sqlConn).Rows)
    {
        if ((row["ReportName"] as string).Equals(reportName))
        {
            isDuplicate = false;
            cmd1 = new SqlCommand("update ReportTable set Report=@Report where ReportName like
@ReportName", sqlConn);
        }
    }
    if (isDuplicate)
    {
        cmd1 = new SqlCommand("insert into ReportTable (ReportName,Report)
Values(@ReportName,@Report)", sqlConn);
    }
    cmd1.Parameters.AddWithValue("@ReportName", reportName);
    cmd1.Parameters.AddWithValue("@Report", report.ToString());
    cmd1.ExecuteNonQuery();
    sqlConn.Close();
}

private string LoadReportFromDB(string reportName)
{
    SqlConnection sqlConn = OpenConnection();
    string report = string.Empty;
    foreach (DataRow row in GetDataTable(sqlConn).Rows)
    {
        if ((row["ReportName"] as string).Equals(reportName))
        {

```

```
report = (string)row["Report"];
break;
}
}
sqlConn.Close();
return report;
}
private List<string> FetchReportListFromDB()
{
    SqlConnection sqlConn = OpenConnection();
    List<string> reportNames = new List<string>();
    foreach (DataRow row in GetDataTable(sqlConn).Rows)
    {
        if (!string.IsNullOrEmpty(row["ReportName"] as string))
        {
            reportNames.Add(row["ReportName"].ToString());
        }
    }
    sqlConn.Close();
    return reportNames;
}
private void RenameReportInDB(string reportName, string renameReport, bool isReportExists)
{
    SqlConnection sqlConn = OpenConnection();
    SqlCommand cmd1 = null;
    if (isReportExists)
    {
        foreach (DataRow row in GetDataTable(sqlConn).Rows)
        {
            if ((row["ReportName"] as string).Equals(reportName))
            {
                cmd1 = new SqlCommand("delete from ReportTable where ReportName like '%" + reportName + "%'",
                    sqlConn);
            }
        }
    }
}
```

```
break;
}
}
cmd1.ExecuteNonQuery();
}
foreach (DataRow row in GetDataTable(sqlConn).Rows)
{
if ((row["ReportName"] as string).Equals(reportName))
{
cmd1 = new SqlCommand("update ReportTable set ReportName=@RenameReport where ReportName
like '%" + reportName + "%'", sqlConn);
break;
}
}
cmd1.Parameters.AddWithValue("@RenameReport", renameReport);
cmd1.ExecuteNonQuery();
sqlConn.Close();
}
private void RemoveReportFromDB(string reportName)
{
SqlConnection sqlConn = OpenConnection();
SqlCommand cmd1 = null;
foreach (DataRow row in GetDataTable(sqlConn).Rows)
{
if ((row["ReportName"] as string).Equals(reportName))
{
cmd1 = new SqlCommand("delete from ReportTable where ReportName like '%" + reportName + "%'",
sqlConn);
break;
}
}
cmd1.ExecuteNonQuery();
sqlConn.Close();
}
```

```

private SqlConnection OpenConnection()
{
    // Replace with your own connection string.
    string connectionString = @"<Enter your valid connection string here>";
    SqlConnection sqlConn = new SqlConnection(connectionString);
    sqlConn.Open();
    return sqlConn;
}

private DataTable GetDataTable(SqlConnection sqlConn)
{
    string xquery = "select * from ReportTable";
    SqlCommand cmd = new SqlCommand(xquery, sqlConn);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
    da.Fill(dt);
    return dt;
}
}

```

5. When you run the app, it will be hosted at <https://localhost:44313>. You can use the hosted URL to save and load reports in the SQL database from the Pivot Table.

Further, let us explore more on how to save, load, rename, delete, and add reports using the built-in toolbar options via Web API controller (aka, PivotController) one-by-one.

[Saving a report](#)

When you select the **“Save a report”** option from the toolbar, the [saveReport](#) event is triggered. In this event, an AJAX request is made to the Web API controller's **SaveReport** method, passing the name of the current report and the current report, which you can use to check and save in the SQL database.

For example, the report shown in the following code snippet will be passed to the **SaveReport** method along with the report name **“Sample Report”** and saved in the SQL database.

```

[App.js]
`js
import React, { Component } from 'react';
import './index.css';
import './App.css';

```

```
import { PivotViewComponent, Inject, FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport,
ConditionalFormatting, NumberFormatting } from '@syncfusion/ej2-react-pivotview';

export default class App extends Component {
  static displayName = App.name;

  dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: this.getPivotData(),
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
  };

  toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove', 'Load',
'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal', 'Formatting', 'FieldList'];

  chartSettings = { title: 'Sales Analysis' };

  gridSettings = { columnWidth: 140 };

  displayOption = { view: 'Both' };

  saveReport(args) {
    var report = JSON.parse(args.report);
    report.dataSourceSettings.dataSource = [];
    fetch('https://localhost:44313/Pivot/SaveReport', {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ reportName: args.reportName, report: JSON.stringify(report) })
    }).then(response => {
      this.fetchReport();
    });
  }

  render() {
```



```

return (<PivotViewComponent id='PivotView' ref={(scope) => { this.pivotObj = scope; }}
dataSourceSettings={this.dataSourceSettings} width={'100%'} height={'450'} showFieldList={true}
gridSettings={this.gridSettings} allowExcelExport={true} allowNumberFormatting={true}
allowConditionalFormatting={true} allowPdfExport={true} showToolbar={true}
allowCalculatedField={true} displayOption={this.displayOption} toolbar={this.toolbarOptions}
newReport={this.newReport.bind(this)} renameReport={this.renameReport.bind(this)}
removeReport={this.removeReport.bind(this)} loadReport={this.loadReport.bind(this)}
fetchReport={this.fetchReport.bind(this)} saveReport={this.saveReport.bind(this)}
toolbarRender={this.beforeToolbarRender.bind(this)} chartSettings={this.chartSettings}>

<Inject services={[FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport, ConditionalFormatting,
NumberFormatting]} />

</PivotViewComponent>;
}
}
,

```

[PivotController.cs]

```

`csharp
namespace MyWebApp.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpPost]
        [Route("Pivot/SaveReport")]
        public void SaveReport([FromBody] Dictionary<string, string> reportArgs)
        {
            SaveReportToDB(reportArgs["reportName"], reportArgs["report"]);
        }
        private void SaveReportToDB(string reportName, string report)
        {
            SqlConnection sqlConn = OpenConnection();
            bool isDuplicate = true;
            SqlCommand cmd1 = null;
            foreach (DataRow row in GetDataTable(sqlConn).Rows)
            {

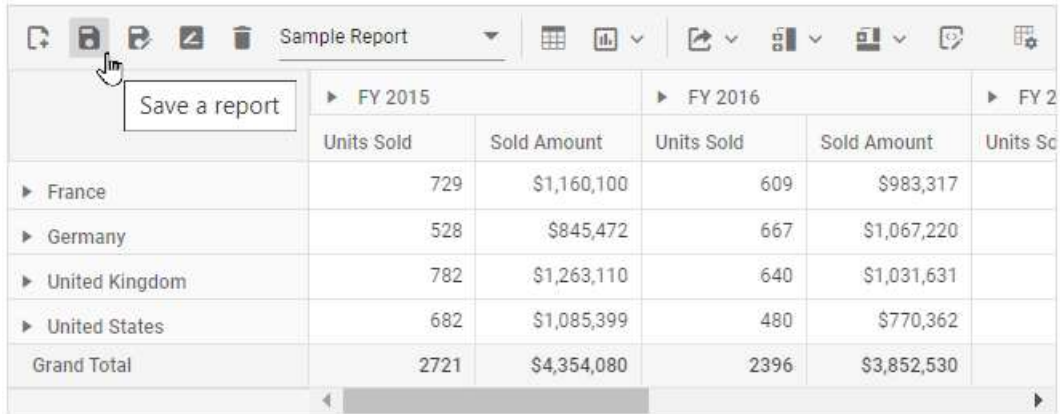
```

```
if ((row["ReportName"] as string).Equals(reportName))
{
    isDuplicate = false;
    cmd1 = new SqlCommand("update ReportTable set Report=@Report where ReportName like
    @ReportName", sqlConn);
}
}
if (isDuplicate)
{
    cmd1 = new SqlCommand("insert into ReportTable (ReportName,Report)
    Values(@ReportName,@Report)", sqlConn);
}
cmd1.Parameters.AddWithValue("@ReportName", reportName);
cmd1.Parameters.AddWithValue("@Report", report.ToString());
cmd1.ExecuteNonQuery();
sqlConn.Close();
}
private SqlConnection OpenConnection()
{
    // Replace with your own connection string.
    string connectionString = @"<Enter your valid connection string here>";
    SqlConnection sqlConn = new SqlConnection(connectionString);
    sqlConn.Open();
    return sqlConn;
}
private DataTable GetDataTable(SqlConnection sqlConn)
{
    string xquery = "select * from ReportTable";
    SqlCommand cmd = new SqlCommand(xquery, sqlConn);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
    da.Fill(dt);
    return dt;
}
```

```

}
}
,

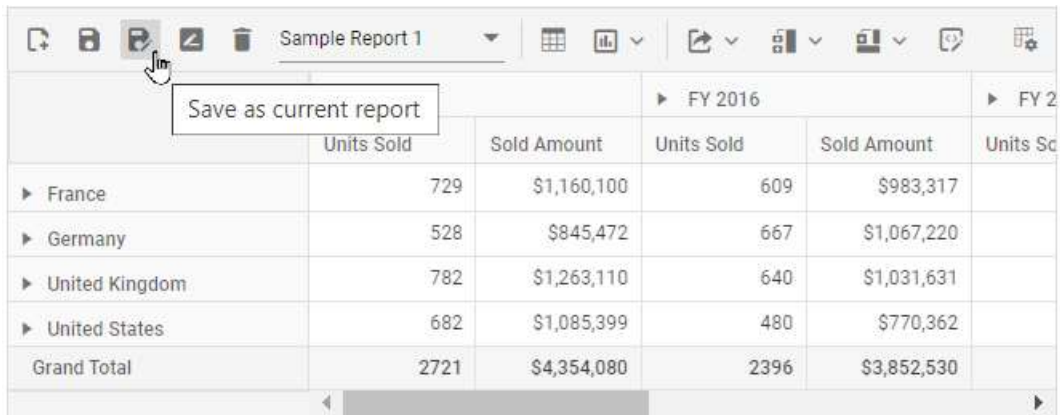
```



The screenshot shows the React Pivotview toolbar with the 'Save a report' button highlighted. A tooltip labeled 'Save a report' is visible. Below the toolbar, a pivot table is displayed with the following data:

	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
France	729	\$1,160,100	609	\$983,317	
Germany	528	\$845,472	667	\$1,067,220	
United Kingdom	782	\$1,263,110	640	\$1,031,631	
United States	682	\$1,085,399	480	\$770,362	
Grand Total	2721	\$4,354,080	2396	\$3,852,530	

In the meantime, you can save a duplicate of the current report to the SQL Server database with a different name by selecting **“Save as current report”** from the toolbar. The [saveReport](#) event will then be triggered with the new report name **“Sample Report 1”** and the current report. You can save them to the SQL Server database after passing them to the Web API service, as mentioned above.



The screenshot shows the React Pivotview toolbar with the 'Save as current report' button highlighted. A tooltip labeled 'Save as current report' is visible. Below the toolbar, a pivot table is displayed with the following data:

	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
France	729	\$1,160,100	609	\$983,317	
Germany	528	\$845,472	667	\$1,067,220	
United Kingdom	782	\$1,263,110	640	\$1,031,631	
United States	682	\$1,085,399	480	\$770,362	
Grand Total	2721	\$4,354,080	2396	\$3,852,530	

Loading a report

When you select the dropdown menu item from the toolbar, the [loadReport](#) event is triggered. In this event, an AJAX request is made to the **LoadReport** method of the Web API controller, passing the name of the selected report. The method uses this information to search for the report in the SQL database, fetch it, and load it into the pivot table.

For example, if the report name **“Sample Report 1”** is selected from a dropdown menu and passed, the **LoadReport** method will use that name to search for the report in the SQL database, retrieve it, and then load it into the pivot table.

[App.js]

```
`js
```

```
import React, { Component } from 'react';
```

```
import './index.css';
```

```
import './App.css';

import { PivotViewComponent, Inject, FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport,
ConditionalFormatting, NumberFormatting } from '@syncfusion/ej2-react-pivotview';

export default class App extends Component {

  static displayName = App.name;

  dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: this.getPivotData(),
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
  };

  toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove', 'Load',
'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal', 'Formatting', 'FieldList'];

  chartSettings = { title: 'Sales Analysis' };

  gridSettings = { columnWidth: 140 };

  displayOption = { view: 'Both' };

  loadReport(args) {
    fetch('https://localhost:44313/Pivot/LoadReport', {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ reportName: args.reportName })
    }).then(res => res.json())
    .then(response => {
      if (response) {
        var report = JSON.parse(response);
        report.dataSourceSettings.dataSource = this.pivotObj.dataSourceSettings.dataSource;
        this.pivotObj.dataSourceSettings = report.dataSourceSettings;
      }
    });
  }
}
```

```

}
});
}
render() {
return (<PivotViewComponent id='PivotView' ref={(scope) => { this.pivotObj = scope; }}
dataSourceSettings={this.dataSourceSettings} width={'100%'} height={'450'} showFieldList={true}
gridSettings={this.gridSettings} allowExcelExport={true} allowNumberFormatting={true}
allowConditionalFormatting={true} allowPdfExport={true} showToolbar={true}
allowCalculatedField={true} displayOption={this.displayOption} toolbar={this.toolbarOptions}
newReport={this.newReport.bind(this)} renameReport={this.renameReport.bind(this)}
removeReport={this.removeReport.bind(this)} loadReport={this.loadReport.bind(this)}
fetchReport={this.fetchReport.bind(this)} saveReport={this.saveReport.bind(this)}
toolbarRender={this.beforeToolbarRender.bind(this)} chartSettings={this.chartSettings}>
<Inject services={[FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport, ConditionalFormatting,
NumberFormatting]} />
</PivotViewComponent>;
}
}
,

```

[PivotController.cs]

```

`csharp
using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using System.Data;
namespace MyWebApp.Controllers
{
[ApiController]
[Route("[controller]")]
public class PivotController : ControllerBase
{
[HttpPost]
[Route("Pivot/LoadReport")]
public IActionResult LoadReport([FromBody] Dictionary<string, string> reportArgs)
{
return Ok((LoadReportFromDB(reportArgs["reportName"])));
}
}

```

```
private string LoadReportFromDB(string reportName)
{
    SqlConnection sqlConn = OpenConnection();
    string report = string.Empty;
    foreach (DataRow row in GetDataTable(sqlConn).Rows)
    {
        if ((row["ReportName"] as string).Equals(reportName))
        {
            report = (string)row["Report"];
            break;
        }
    }
    sqlConn.Close();
    return report;
}

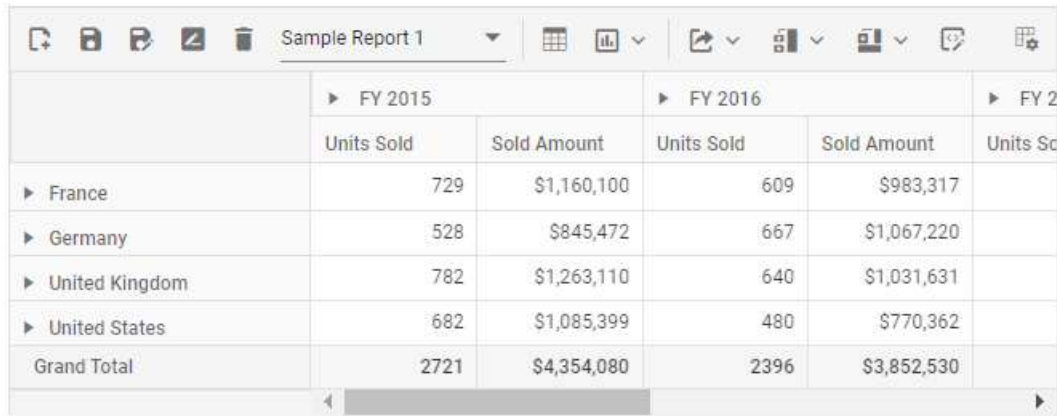
private SqlConnection OpenConnection()
{
    // Replace with your own connection string.
    string connectionString = @"<Enter your valid connection string here>";
    SqlConnection sqlConn = new SqlConnection(connectionString);
    sqlConn.Open();
    return sqlConn;
}

private DataTable GetDataTable(SqlConnection sqlConn)
{
    string xquery = "select * from ReportTable";
    SqlCommand cmd = new SqlCommand(xquery, sqlConn);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
    da.Fill(dt);
    return dt;
}
}
```

```

}
,

```



	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
France	729	\$1,160,100	609	\$983,317	
Germany	528	\$845,472	667	\$1,067,220	
United Kingdom	782	\$1,263,110	640	\$1,031,631	
United States	682	\$1,085,399	480	\$770,362	
Grand Total	2721	\$4,354,080	2396	\$3,852,530	

Renaming a report

When you select the **“Rename a current report”** option from the toolbar, the [renameReport](#) event is triggered. In this event, an AJAX request is made to the **RenameReport** method of the Web API controller, passing the current and new report names, where you can use the current report name to identify the report and resave it with the new report name in the SQL database.

For example, if we rename the current report from **“Sample Report 1”** to **“Sample Report 2”**, both **“Sample Report 1”** and **“Sample Report 2”** will be passed to the **RenameReport** method, which will rename the current report with the new report name **“Sample Report 2”** in the SQL database.

[App.js]

```
`js
```

```
import React, { Component } from 'react';
```

```
import './index.css';
```

```
import './App.css';
```

```
import { PivotViewComponent, Inject, FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport, ConditionalFormatting, NumberFormatting } from '@syncfusion/ej2-react-pivotview';
```

```
export default class App extends Component {
```

```
  static displayName = App.name;
```

```
  dataSourceSettings = {
```

```
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
```

```
    dataSource: this.getPivotData(),
```

```
    expandAll: false,
```

```
    filters: [],
```

```
    formatSettings: [{ name: 'Amount', format: 'C0' }],
```

```
    rows: [{ name: 'Country' }, { name: 'Products' }],
```

```
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
```

```

};
toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove', 'Load',
'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal', 'Formatting', 'FieldList'];
chartSettings = { title: 'Sales Analysis' };
gridSettings = { columnWidth: 140 };
displayOption = { view: 'Both' };
renameReport(args) {
fetch('https://localhost:44313/Pivot/RenameReport', {
method: 'POST',
headers: {
'Accept': 'application/json',
'Content-Type': 'application/json',
},
body: JSON.stringify({ reportName: args.reportName, renameReport: args.rename, isReportExists:
args.isReportExists })
}).then(response => {
this.fetchReport();
});
}
render() {
return (<PivotViewComponent id='PivotView' ref={(scope) => { this.pivotObj = scope; }}
dataSourceSettings={this.dataSourceSettings} width={'100%'} height={'450'} showFieldList={true}
gridSettings={this.gridSettings} allowExcelExport={true} allowNumberFormatting={true}
allowConditionalFormatting={true} allowPdfExport={true} showToolbar={true}
allowCalculatedField={true} displayOption={this.displayOption} toolbar={this.toolbarOptions}
newReport={this.newReport.bind(this)} renameReport={this.renameReport.bind(this)}
removeReport={this.removeReport.bind(this)} loadReport={this.loadReport.bind(this)}
fetchReport={this.fetchReport.bind(this)} saveReport={this.saveReport.bind(this)}
toolbarRender={this.beforeToolbarRender.bind(this)} chartSettings={this.chartSettings}>
<Inject services={[FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport, ConditionalFormatting,
NumberFormatting]} />
</PivotViewComponent>);
}
}
,

```

[PivotController.cs]


```
`csharp
using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using System.Data;
namespace MyWebApp.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpPost]
        [Route("Pivot/RenameReport")]
        public void RenameReport([FromBody] RenameReportDB reportArgs)
        {
            RenameReportInDB(reportArgs.ReportName, reportArgs.RenameReport, reportArgs.isReportExists);
        }
        public class RenameReportDB
        {
            public string ReportName { get; set; }
            public string RenameReport { get; set; }
            public bool isReportExists { get; set; }
        }
        private void RenameReportInDB(string reportName, string renameReport, bool isReportExists)
        {
            SqlConnection sqlConn = OpenConnection();
            SqlCommand cmd1 = null;
            if (isReportExists)
            {
                foreach (DataRow row in GetDataTable(sqlConn).Rows)
                {
                    if ((row["ReportName"] as string).Equals(reportName))
                    {

```

```

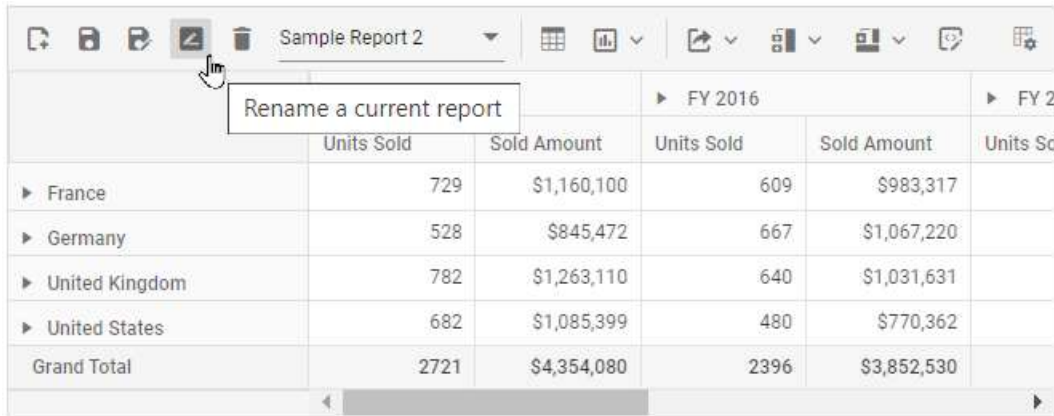
cmd1 = new SqlCommand("delete from ReportTable where ReportName like '%" + reportName + "%'",
sqlConn);
break;
}
}
cmd1.ExecuteNonQuery();
}
foreach (DataRow row in GetDataTable(sqlConn).Rows)
{
if ((row["ReportName"] as string).Equals(reportName))
{
cmd1 = new SqlCommand("update ReportTable set ReportName=@RenameReport where ReportName
like '%" + reportName + "%'", sqlConn);
break;
}
}
cmd1.Parameters.AddWithValue("@RenameReport", renameReport);
cmd1.ExecuteNonQuery();
sqlConn.Close();
}
private SqlConnection OpenConnection()
{
// Replace with your own connection string.
string connectionString = @"<Enter your valid connection string here>";
SqlConnection sqlConn = new SqlConnection(connectionString);
sqlConn.Open();
return sqlConn;
}
private DataTable GetDataTable(SqlConnection sqlConn)
{
string xquery = "select * from ReportTable";
SqlCommand cmd = new SqlCommand(xquery, sqlConn);
SqlDataAdapter da = new SqlDataAdapter(cmd);
DataTable dt = new DataTable();

```

```

da.Fill(dt);
return dt;
}
}
}
,

```



	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
France	729	\$1,160,100	609	\$983,317	
Germany	528	\$845,472	667	\$1,067,220	
United Kingdom	782	\$1,263,110	640	\$1,031,631	
United States	682	\$1,085,399	480	\$770,362	
Grand Total	2721	\$4,354,080	2396	\$3,852,530	

Deleting a report

When you select the **“Delete a current report”** option from the toolbar, the [removeReport](#) event is triggered. In this event, an AJAX request is made to the **RemoveReport** method of the Web API controller, passing the current report name to identify and delete the appropriate report from the SQL database.

Note: * If the current report **n** from the pivot table is deleted, the pivot table will automatically load the last report from the report list.

* When a report is removed from a pivot table with only one report, the SQL database refreshes; however, the pivot table will continue to show the removed report until a new report is added to the pivot table.

For example, if we delete the current report **“Sample Report 2”** from the pivot table, the current report name **“Sample Report 2”** is passed to the **RemoveReport** method, which allows you to identify and delete the report from the SQL database.

[App.js]

```

`js
import React, { Component } from 'react';
import './index.css';
import './App.css';

import { PivotViewComponent, Inject, FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport,
ConditionalFormatting, NumberFormatting } from '@syncfusion/ej2-react-pivotview';

export default class App extends Component {
static displayName = App.name;

```

```

dataSourceSettings = {
  columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
  dataSource: this.getPivotData(),
  expandAll: false,
  filters: [],
  formatSettings: [{ name: 'Amount', format: 'C0' }],
  rows: [{ name: 'Country' }, { name: 'Products' }],
  values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
};

toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove', 'Load',
  'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal', 'Formatting', 'FieldList'];
chartSettings = { title: 'Sales Analysis' };
gridSettings = { columnWidth: 140 };
displayOption = { view: 'Both' };
removeReport(args) {
  fetch('https://localhost:44313/Pivot/RemoveReport', {
    method: 'POST',
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ reportName: args.reportName })
  }).then(response => {
    this.fetchReport();
  });
}

render() {
  return (<PivotViewComponent id='PivotView' ref={(scope) => { this.pivotObj = scope; }}
    dataSourceSettings={this.dataSourceSettings} width={'100%'} height={'450'} showFieldList={true}
    gridSettings={this.gridSettings} allowExcelExport={true} allowNumberFormatting={true}
    allowConditionalFormatting={true} allowPdfExport={true} showToolbar={true}
    allowCalculatedField={true} displayOption={this.displayOption} toolbar={this.toolbarOptions}
    newReport={this.newReport.bind(this)} renameReport={this.renameReport.bind(this)}
    removeReport={this.removeReport.bind(this)} loadReport={this.loadReport.bind(this)}
    fetchReport={this.fetchReport.bind(this)} saveReport={this.saveReport.bind(this)}
    toolbarRender={this.beforeToolbarRender.bind(this)} chartSettings={this.chartSettings}>

```

```
<Inject services={{FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport, ConditionalFormatting,
NumberFormatting}} />
```

```
</PivotViewComponent>;
```

```
}
```

```
}
```

```
,
```

```
[PivotController.cs]
```

```
`csharp
```

```
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.Data.SqlClient;
```

```
using System.Data;
```

```
namespace MyWebApp.Controllers
```

```
{
```

```
[ApiController]
```

```
[Route("[controller]")]
```

```
public class PivotController : ControllerBase
```

```
{
```

```
[HttpPost]
```

```
[Route("Pivot/RemoveReport")]
```

```
public void RemoveReport([FromBody] Dictionary<string, string> reportArgs)
```

```
{
```

```
RemoveReportFromDB(reportArgs["reportName"]);
```

```
}
```

```
private void RemoveReportFromDB(string reportName)
```

```
{
```

```
SqlConnection sqlConn = OpenConnection();
```

```
SqlCommand cmd1 = null;
```

```
foreach (DataRow row in GetDataTable(sqlConn).Rows)
```

```
{
```

```
if ((row["ReportName"] as string).Equals(reportName))
```

```
{
```

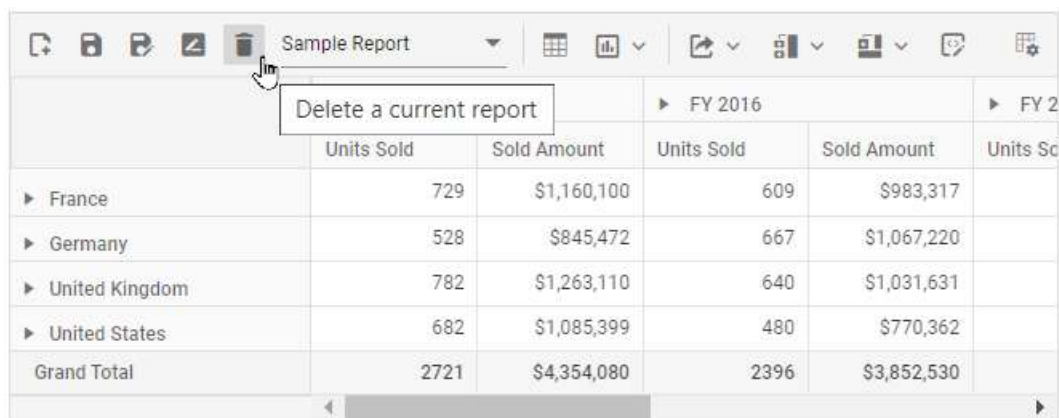
```
cmd1 = new SqlCommand("delete from ReportTable where ReportName like '%" + reportName + "%'",
sqlConn);
```

```
break;
```

```

}
}
cmd1.ExecuteNonQuery();
sqlConn.Close();
}
private SqlConnection OpenConnection()
{
// Replace with your own connection string.
string connectionString = @"<Enter your valid connection string here>";
SqlConnection sqlConn = new SqlConnection(connectionString);
sqlConn.Open();
return sqlConn;
}
private DataTable GetDataTable(SqlConnection sqlConn)
{
string xquery = "select * from ReportTable";
SqlCommand cmd = new SqlCommand(xquery, sqlConn);
SqlDataAdapter da = new SqlDataAdapter(cmd);
DataTable dt = new DataTable();
da.Fill(dt);
return dt;
}
}
}
,

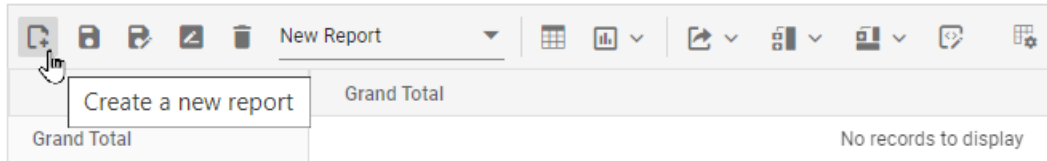
```



	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
▶ France	729	\$1,160,100	609	\$983,317	
▶ Germany	528	\$845,472	667	\$1,067,220	
▶ United Kingdom	782	\$1,263,110	640	\$1,031,631	
▶ United States	682	\$1,085,399	480	\$770,362	
Grand Total	2721	\$4,354,080	2396	\$3,852,530	

Adding a report

When you select the “**Create a new report**” option from the toolbar, the [newReport](#) event is triggered, followed by the [saveReport](#) event. To save this new report to the SQL database, use the [saveReport](#) event triggered later, and then follow the save report briefing in the preceding [topic](#).



Limitations with respect to report manipulation

Below points need to be considered when saving the report to SQL Server database.

- **Data source:** Both raw data and aggregated data won't be saved and loaded from the database.
- **Hyperlinks:** Option to link external facts via pivot table cells won't be saved and loaded from the database.
- The pivot table should always load reports from the SQL database based on the data source that is currently bound to it.

In [this](#) GitHub repository, you can find our React Pivot Table sample and ASP.NET Core Web Application to save and load reports from SQL Server database.

Events

FetchReport

The event [fetchReport](#) is triggered when dropdown list is clicked in the toolbar in-order to retrieve and populate saved reports. It has following parameter - `reportName`. This event allows user to fetch the report names from local storage and populate the dropdown list.

LoadReport

The event [loadReport](#) is triggered when a report is selected from the dropdown list in the toolbar. It has following parameters - `report` and `reportName`. This event allows user to load the selected report to the pivot table.

NewReport

The event [newReport](#) is triggered when the new report icon is clicked in the toolbar. It has following parameter - `report`. This event allows user to create new report and add to the report list.

RenameReport

The event [renameReport](#) is triggered when rename report icon is clicked in the toolbar. It has following parameters - `rename`, `report` and `reportName`. This event allows user to rename the selected report from the report list.

RemoveReport

The event [removeReport](#) is triggered when remove report icon is clicked in the toolbar. It has following parameters - `report` and `reportName`. This event allows user to remove the selected report from the report list.

SaveReport

The event [saveReport](#) is triggered when save report icon is clicked in the toolbar. It has following parameters - `report` and `reportName`. This event allows user to save the altered report to the report list.

ToolbarRender

The [toolbarRender](#) event is triggered when the toolbar is rendered. It has the `customToolbar` parameter. This event helps to customize the built-in toolbar items and to [include new toolbar item\(s\)](#).

INDEX.JSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent, Inject, Toolbar, FieldList } from
'@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    let toolbarOptions = ["Save", "Export", "FieldList"];
    function saveReport(args) {
        let reports = [];
        let isSaved = false;
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!== "") {
            reports = JSON.parse(localStorage.pivotviewReports);
        }
        if (args.report && args.reportName && args.reportName !== '') {
            reports.map(function (item) {
                if (args.reportName === item.reportName) {
                    item.report = args.report;
                    isSaved = true;
                }
            });
            if (!isSaved) {
                reports.push(args);
            }
            localStorage.pivotviewReports = JSON.stringify(reports);
        }
    }
    function beforeToolbarRender(args) {
        args.customToolbar.splice(2, 0, {
            prefixIcon: 'e-rename-report e-icons', tooltipText: 'Custom
Button',
            click: customButton.bind(this),
        });
        args.customToolbar.splice(3, 0, {
            prefixIcon: 'e-tool-expand e-icons', tooltipText:
'Expand/Collapse',
            click: toolbarClicked.bind(this),
        });
        args.customToolbar[0].align = "Left";
    }
}
```



```

        args.customToolbar[1].align = "Center";
        args.customToolbar[2].align = "Right";
    }
    function customButton(args) {
        // Here you can customize the click event for custom button
    }
    function toolbarClicked(args) {
        pivotObj.dataSourceSettings.expandAll =
!pivotObj.dataSourceSettings.expandAll;
    }
    return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
gridSettings={{ columnWidth: 140 }} showToolbar={true} displayOption={{
view: 'Both' }} showFieldList={true} toolbar={toolbarOptions}
toolbarRender={beforeToolbarRender.bind(this)}><Inject services={[Toolbar,
FieldList]} saveReport={saveReport.bind(this)}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    PivotViewComponent, IDataOptions, Inject, Toolbar, ToolbarArgs,
    SaveReportArgs, FieldList
} from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    };
    let pivotObj: PivotViewComponent;
    let toolbarOptions: any = [ "Save", "Export", "FieldList"];
    function saveReport(args: any): void {
        let reports: SaveReportArgs[] = [];
        let isSaved: boolean = false;
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!== "") {
            reports = JSON.parse(localStorage.pivotviewReports);
        }
        if (args.report && args.reportName && args.reportName !== '') {
            reports.map(function (item: any): any {
                if (args.reportName === item.reportName) {
                    item.report = args.report; isSaved = true;

```

```

    });
    if (!isSaved) {
        reports.push(args);
    }
    localStorage.pivotviewReports = JSON.stringify(reports);
}
}
function beforeToolbarRender(args: ToolbarArgs): void {
    args.customToolbar.splice(2, 0, {
        prefixIcon: 'e-rename-report e-icons', tooltipText: 'Custom
Button',
        click: customButton.bind(this),
    });
    args.customToolbar.splice(3, 0, {
        prefixIcon: 'e-tool-expand e-icons', tooltipText:
'Expand/Collapse',
        click: toolbarClicked.bind(this),
    });
    args.customToolbar[0].align = "Left";
    args.customToolbar[1].align = "Center";
    args.customToolbar[2].align = "Right";
}
function customButton(args: any): void {
    // Here you can customize the click event for custom button
}
function toolbarClicked(args: any): void {
    pivotObj.dataSourceSettings.expandAll =
!pivotObj.dataSourceSettings.expandAll;
}
return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
gridSettings={{ columnWidth: 140 }} showToolbar={true} displayOption={{
view: 'Both' }} showFieldList={true} toolbar={toolbarOptions}
toolbarRender={beforeToolbarRender.bind(this)}><Inject services={[Toolbar,
FieldList]} saveReport={saveReport.bind(this)} /></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

BeforeExport

The pivot table (or) pivot chart can be exported as a pdf, excel, csv etc., document using the toolbar options. And, you can customize the export settings for exporting document by using the [beforeExport](#) event in the toolbar.

For example, you can add the header and footer for the pdf document by setting the **header** and **footer** properties for the **pdfExportProperties** in the [beforeExport](#) event.

INDEX.JSX

```

{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```

import { PivotViewComponent, Inject, FieldList, CalculatedField, Toolbar,
PDFExport, ExcelExport, ConditionalFormatting } from '@syncfusion/ej2-react-
pivotview';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  let toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
  'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'FieldList'];
  function saveReport(args) {
    let reports = [];
    let isSaved = false;
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
!== "") {
      reports = JSON.parse(localStorage.pivotviewReports);
    }
    if (args.report && args.reportName && args.reportName !== '') {
      reports.map(function (item) {
        if (args.reportName === item.reportName) {
          item.report = args.report;
          isSaved = true;
        }
      });
    }
    if (!isSaved) {
      reports.push(args);
    }
    localStorage.pivotviewReports = JSON.stringify(reports);
  }
  function fetchReport(args) {
    let reportCollection = [];
    let reeportList = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
!== "") {
      reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item) {
      reeportList.push(item.reportName);
    });
    args.reportName = reeportList;
  }
  function loadReport(args) {
    let reportCollection = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
!== "") {
      reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
  }
}

```

```

    }
    reportCollection.map(function (item) {
        if (args.reportName === item.reportName) {
            args.report = item.report;
        }
    });
    if (args.report) {
        pivotObj.dataSource = JSON.parse(args.report).dataSource;
    }
}
function removeReport(args) {
    let reportCollection = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
    !== "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    for (let i = 0; i < reportCollection.length; i++) {
        if (reportCollection[i].reportName === args.reportName) {
            reportCollection.splice(i, 1);
        }
    }
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
    !== "") {
        localStorage.pivotviewReports =
        JSON.stringify(reportCollection);
    }
}
function renameReport(args) {
    let reportCollection = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
    !== "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item) { if (args.reportName ===
    item.reportName) {
        item.reportName = args.rename;
    } });
    if (localStorage.pivotviewReports && localStorage.pivotviewReports
    !== "") {
        localStorage.pivotviewReports =
        JSON.stringify(reportCollection);
    }
}
function newReport() {
    pivotObj.setProperties({ dataSource: { columns: [], rows: [],
    values: [], filters: [] } }, false);
}
function beforeExport(args) {
    args.excelExportProperties = {
        header: {
            headerRows: 2,
            rows: [
                { cells: [{ colSpan: 4, value: "Pivot Table", style: {
    fontColor: '#C67878', fontSize: 20, hAlign: 'Center', bold: true, underline:
    true } }]} ]
            }
        },
    },

```

```

        footer: {
            footerRows: 4,
            rows: [
                { cells: [{ colSpan: 4, value: "Thank you for your business!", style: { hAlign: 'Center', bold: true } }] },
                { cells: [{ colSpan: 4, value: "!Visit Again!", style: { hAlign: 'Center', bold: true } }] }
            ]
        }
    };
    args.pdfExportProperties = {
        header: {
            fromTop: 0,
            height: 130,
            contents: [
                {
                    type: 'Text',
                    value: "Pivot Table",
                    position: { x: 0, y: 50 },
                    style: { textBrushColor: '#000000', fontSize: 13,
dashStyle: 'Solid', hAlign: 'Center' }
                }
            ]
        },
        footer: {
            fromBottom: 160,
            height: 150,
            contents: [
                {
                    type: 'PageNumber',
                    pageNumberType: 'Arabic',
                    format: 'Page { $current } of { $total }',
                    position: { x: 0, y: 25 },
                    style: { textBrushColor: '#02007a', fontSize: 15 }
                }
            ]
        }
    };
}

return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
showFieldList={true} gridSettings={{ columnWidth: 140 }}
allowExcelExport={true} allowConditionalFormatting={true}
allowPdfExport={true} showToolbar={true} allowCalculatedField={true}
displayOption={{ view: 'Both' }} toolbar={toolbarOptions}
newReport={newReport.bind(this)} renameReport={renameReport.bind(this)}
removeReport={removeReport.bind(this)} loadReport={loadReport.bind(this)}
fetchReport={fetchReport.bind(this)} saveReport={saveReport.bind(this)}
beforeExport={beforeExport.bind(this)}><Inject services={[FieldList,
CalculatedField, Toolbar, PDFExport, ExcelExport,
ConditionalFormatting]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% enddraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    PivotViewComponent, IDataOptions, Inject, FieldList, CalculatedField,
    Toolbar, PDFExport, ExcelExport, ConditionalFormatting, SaveReportArgs,
    FetchReportArgs, LoadReportArgs, RemoveReportArgs, RenameReportArgs,
    BeforeExportEventArgs
} from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj: PivotViewComponent;
    let toolbarOptions: any = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
    'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'FieldList'];
    function saveReport(args: any): void {
        let reports: SaveReportArgs[] = [];
        let isSaved: boolean = false;
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!== "") {
            reports = JSON.parse(localStorage.pivotviewReports);
        }
        if (args.report && args.reportName && args.reportName !== '') {
            reports.map(function (item: any): any {
                if (args.reportName === item.reportName) {
                    item.report = args.report; isSaved = true;
                }
            });
            if (!isSaved) {
                reports.push(args);
            }
            localStorage.pivotviewReports = JSON.stringify(reports);
        }
    }
    function fetchReport(args: FetchReportArgs): void {
        let reportCollection: string[] = [];
        let reeportList: string[] = [];
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!== "") {
            reportCollection = JSON.parse(localStorage.pivotviewReports);
        }
    }
}

```

```

        reportCollection.map(function (item: any): void {
reeporList.push(item.reportName); });
        args.reportName = reeporList;
    }
    function loadReport(args: LoadReportArgs): void {
        let reportCollection: string[] = [];
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= "" ) {
            reportCollection = JSON.parse(localStorage.pivotviewReports);
        }
        reportCollection.map(function (item: any): void {
            if (args.reportName === item.reportName) {
                args.report = item.report;
            }
        });
        if (args.report) {
            pivotObj.dataSource = JSON.parse(args.report).dataSource;
        }
    }
    function removeReport(args: RemoveReportArgs): void {
        let reportCollection: any[] = [];
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= "" ) {
            reportCollection = JSON.parse(localStorage.pivotviewReports);
        }
        for (let i: number = 0; i < reportCollection.length; i++) {
            if (reportCollection[i].reportName === args.reportName) {
                reportCollection.splice(i, 1);
            }
        }
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= "" ) {
            localStorage.pivotviewReports =
JSON.stringify(reportCollection);
        }
    }
    function renameReport(args: RenameReportArgs): void {
        let reportCollection: string[] = [];
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= "" ) {
            reportCollection = JSON.parse(localStorage.pivotviewReports);
        }
        reportCollection.map(function (item: any): any { if (args.reportName
=== item.reportName) { item.reportName = args.rename; } });
        if (localStorage.pivotviewReports && localStorage.pivotviewReports
!= "" ) {
            localStorage.pivotviewReports =
JSON.stringify(reportCollection);
        }
    }
    function newReport(): void {
        pivotObj.setProperties({ dataSource: { columns: [], rows: [],
values: [], filters: [] } }, false);
    }
    function beforeExport(args: BeforeExportEventArgs): void {
        args.excelExportProperties = {
            header: {

```

```

        headerRows: 2,
        rows: [
            { cells: [{ colSpan: 4, value: "Pivot Table", style:
{ fontColor: '#C67878', fontSize: 20, hAlign: 'Center', bold: true,
underline: true } }] }
        ]
    },
    footer: {
        footerRows: 4,
        rows: [
            { cells: [{ colSpan: 4, value: "Thank you for your
business!", style: { hAlign: 'Center', bold: true } }] },
            { cells: [{ colSpan: 4, value: "!Visit Again!",
style: { hAlign: 'Center', bold: true } }] }
        ]
    }
};
args.pdfExportProperties = {
    header: {
        fromTop: 0,
        height: 130,
        contents: [
            {
                type: 'Text',
                value: "Pivot Table",
                position: { x: 0, y: 50 },
                style: { textBrushColor: '#000000', fontSize:
13, dashStyle:'Solid',hAlign:'Center' }
            }
        ]
    },
    footer: {
        fromBottom: 160,
        height: 150,
        contents: [
            {
                type: 'PageNumber',
                pageNumberType: 'Arabic',
                format: 'Page { $current } of { $total }',
                position: { x: 0, y: 25 },
                style: { textBrushColor: '#02007a', fontSize: 15
}
            }
        ]
    }
};
}

```

```

    return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={ '100%' } height={350}
showFieldList={true} gridSettings={{ columnWidth: 140 }}
allowExcelExport={true} allowConditionalFormatting={true}
allowPdfExport={true} showToolBar={true} allowCalculatedField={true}
displayOption={{ view: 'Both' }} toolbar={toolbarOptions}
newReport={newReport.bind(this)} renameReport={renameReport.bind(this)}
removeReport={removeReport.bind(this)} loadReport={loadReport.bind(this)}
fetchReport={fetchReport.bind(this)} saveReport={saveReport.bind(this)}

```



```
beforeExport={beforeExport.bind(this)}><Inject services={[FieldList,
CalculatedField, Toolbar, PDFExport, ExcelExport, ConditionalFormatting]}
/></PivotViewComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}
```

ActionBegin

The event [actionBegin](#) triggers when the UI actions such as switching between pivot table and pivot chart, changing chart types, conditional formatting, exporting, etc. that are present in toolbar UI begin. This allows user to identify the current action being performed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action began. The following are the UI actions and their names:

Action	Action Name
-----	-----
New report	Add new report
Save report	Save current report
Save as report	Save as current report
Rename report	Rename current report
Remove report	Remove current report
Report change	Report change
Conditional Formatting	Open conditional formatting dialog
Number Formatting	Open number formatting dialog
Export menu	PDF export, Excel export, CSV export
Show Fieldlist	Open field list
Show Table	Show table view
Chart menu	Show chart view
Sub-totals menu	Hide sub-totals, Show row sub-totals, Show column sub-totals, Show sub-totals
Grand totals menu	Hide grand totals, Show row grand totals, Show column grand totals, Show grand totals

- **cancel**: It allows user to restrict the current action.

In the below sample, toolbar UI actions such as add new report and save current report can be restricted by setting the **args.cancel** option to **true** in the **actionBegin** event.

INDEX.JSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent, Inject, FieldList, CalculatedField, Toolbar,
PDFExport, ExcelExport, ConditionalFormatting, NumberFormatting } from
'@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' } ]
  };
  let pivotObj;
  let toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
  'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'];
  return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
actionBegin={actionBegin.bind(this)} showFieldList={true} gridSettings={{
columnWidth: 140 }} allowExcelExport={true}
allowConditionalFormatting={true} allowNumberFormatting={true}
allowPdfExport={true} showToolbar={true} allowCalculatedField={true}
displayOption={{ view: 'Both' }} toolbar={toolbarOptions}><Inject
services={[FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport,
ConditionalFormatting, NumberFormatting]}/></PivotViewComponent>);
  function actionBegin(args) {
    if (args.actionName == 'Add new report' || args.actionName == 'Save
current report') {
      args.cancel = true;
    }
  }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  PivotViewComponent, IDataOptions, Inject, FieldList, CalculatedField,
  Toolbar, PDFExport, ExcelExport, ConditionalFormatting, SaveReportArgs,
```

```

    FetchReportArgs, LoadReportArgs, RemoveReportArgs, RenameReportArgs,
    NumberFormatting, PivotActionBeginEventArgs
} from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj: PivotViewComponent;
    let toolbarOptions: any = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
    'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'];

    return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
actionBegin={actionBegin.bind(this)} showFieldList={true} gridSettings={{
columnWidth: 140 }} allowExcelExport={true}
allowConditionalFormatting={true} allowNumberFormatting={true}
allowPdfExport={true} showToolbar={true} allowCalculatedField={true}
displayOption={{ view: 'Both' }} toolbar={toolbarOptions}><Inject
services={[FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport,
ConditionalFormatting, NumberFormatting]} /></PivotViewComponent>);
    function actionBegin(args: PivotActionBeginEventArgs): void {
        if (args.actionName == 'Add new report' || args.actionName == 'Save
current report') {
            args.cancel = true;
        }
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

ActionComplete

The event [actionComplete](#) triggers when the UI actions such as switching between pivot table and pivot chart, changing chart types, conditional formatting, exporting, etc. that are present in toolbar UI, is completed. This allows user to identify the current UI actions being completed at runtime. It has the following parameters:

- **dataSourceSettings:** It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName:** It holds the name of the current action completed. The following are the UI actions and their names:

| Action | Action Name |

|-----|-----|

| New report | New report added |

| Save report | Report saved |

| Save as report | Report re-saved |

| Rename report | Report renamed |

| Remove report | Report removed |

| Report change | Report changed |

| Conditional Formatting | Conditionally formatted |

| Number Formatting | Number formatted |

| Export menu | PDF exported, Excel exported, CSV exported |

| Show Fieldlist | Field list closed |

| Show Table | Table view shown |

| Sub-totals menu | Sub-totals hidden, Row sub-totals shown, Column sub-totals shown, Sub-totals shown |

| Grand totals menu | Grand totals hidden, Row grand totals shown, Column grand totals shown, Grand totals shown |

- **actionInfo**: It holds the unique information about the current UI action. For example, while adding new report, the event argument contains information such as report name and the action name.

INDEX.JSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent, Inject, FieldList, CalculatedField, Toolbar,
PDFExport, ExcelExport, ConditionalFormatting, NumberFormatting } from
'@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
```

```

    let toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
    'Load',
    'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
    'ConditionalFormatting', 'NumberFormatting', 'FieldList'];
    return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
    dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
    showFieldList={true} gridSettings={{ columnWidth: 140 }}
    allowExcelExport={true} allowConditionalFormatting={true}
    allowNumberFormatting={true} allowPdfExport={true} showToolbar={true}
    allowCalculatedField={true} displayOption={{ view: 'Both' }}
    actionComplete={actionComplete.bind(this)} toolbar={toolbarOptions}><Inject
    services={[FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport,
    ConditionalFormatting, NumberFormatting]}/></PivotViewComponent>);
    function actionComplete(args) {
        if (args.actionName == 'New report added' || args.actionName ==
    'Report saved') {
            // Triggers when the toolbar UI actions such as add new report
    and save current report icon are completed.
        }
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    PivotViewComponent, IDataOptions, Inject, FieldList, CalculatedField,
    Toolbar, PDFExport, ExcelExport, ConditionalFormatting, SaveReportArgs,
    FetchReportArgs, LoadReportArgs, RemoveReportArgs, RenameReportArgs,
    NumberFormatting, PivotActionCompleteEventArgs
} from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
    'Quarter' }],
        dataSource: pivotData as IDataSet[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
    caption: 'Sold Amount' } ]
    };
    let pivotObj: PivotViewComponent;
    let toolbarOptions: any = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
    'Load',
    'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
    'ConditionalFormatting', 'NumberFormatting', 'FieldList'];

```

```

    return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
    dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
    showFieldList={true} gridSettings={{ columnWidth: 140 }}
    allowExcelExport={true} allowConditionalFormatting={true}
    allowNumberFormatting={true} allowPdfExport={true} showToolbar={true}
    allowCalculatedField={true} displayOption={{ view: 'Both' }}
    actionComplete={actionComplete.bind(this)} toolbar={toolbarOptions}><Inject
    services={[FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport,
    ConditionalFormatting, NumberFormatting]} /></PivotViewComponent>);
    function actionComplete(args: PivotActionCompleteEventArgs): void {
        if (args.actionName == 'New report added' || args.actionName ==
        'Report saved') {
            // Triggers when the toolbar UI actions such as add new report
            and save current report icon are completed.
        }
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% enddraw %}

```

ActionFailure

The event [actionFailure](#) triggers when the current UI action fails to achieve the desired result. It has the following parameters:

- **actionName**: It holds the name of the current action failed. The following are the UI actions and their names:

Action	Action Name
-----	-----
New report	Add new report
Save report	Save current report
Save as report	Save as current report
Rename report	Rename current report
Remove report	Remove current report
Report change	Report change
Conditional Formatting	Open conditional formatting dialog
Number Formatting	Open number formatting dialog
Export menu	PDF export, Excel export, CSV export
Show Fieldlist	Open field list
Show Table	Show table view
Chart menu	Show chart view
Sub-totals menu	Hide sub-totals, Show row sub-totals, Show column sub-totals, Show sub-totals

| Grand totals menu | Hide grand totals, Show row grand totals, Show column grand totals, Show grand totals |

- **errorInfo**: It holds the error information of the current UI action.

INDEX.JSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent, Inject, FieldList, CalculatedField, Toolbar,
PDFExport, ExcelExport, ConditionalFormatting, NumberFormatting } from
'@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  let toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
  'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'];
  return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
showFieldList={true} gridSettings={{ columnWidth: 140 }}
allowExcelExport={true} allowConditionalFormatting={true}
allowNumberFormatting={true} allowPdfExport={true} showToolbar={true}
allowCalculatedField={true} displayOption={{ view: 'Both' }}
actionFailure={actionFailure.bind(this)} toolbar={toolbarOptions}><Inject
services={[FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport,
ConditionalFormatting, NumberFormatting]}/></PivotViewComponent>);
  function actionFailure(args) {
    if (args.actionName == 'Add new report' || args.actionName == 'Save
current report') {
      // Triggers when the current UI action fails to achieve the
desired result.
    }
  }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    PivotViewComponent, IDataOptions, Inject, FieldList, CalculatedField,
    Toolbar, PDFExport, ExcelExport, ConditionalFormatting, SaveReportArgs,
    FetchReportArgs, LoadReportArgs, RemoveReportArgs, RenameReportArgs,
    NumberFormatting, PivotActionFailureEventArgs
} from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj: PivotViewComponent;
    let toolbarOptions: any = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
    'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'];

    return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'} height={350}
showFieldList={true} gridSettings={{ columnWidth: 140 }}
allowExcelExport={true} allowConditionalFormatting={true}
allowNumberFormatting={true} allowPdfExport={true} showToolbar={true}
allowCalculatedField={true} displayOption={{ view: 'Both' }}
actionFailure={actionFailure.bind(this)} toolbar={toolbarOptions}><Inject
services={[FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport,
ConditionalFormatting, NumberFormatting]} /></PivotViewComponent>);
    function actionFailure(args: PivotActionFailureEventArgs): void {
        if (args.actionName == 'Add new report' || args.actionName == 'Save
current report') {
            // Triggers when the current UI action fails to achieve the
desired result.
        }
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}
```

See Also

- [Toolbar Component](#)

Tool tip in React Pivotview component

The tooltip can be enabled or disabled by setting the [showTooltip](#) property to **true**. By default, tooltip is enabled in the pivot table.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} showTooltip={true}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataSet, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataSet[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
```

```

return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} showTooltip={true}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Tooltip Template

User can design their own tooltip by setting the property [tooltipTemplate](#) with own HTML elements. The property accepts both HTML string and ID attribute. The following place holders are available to display its dynamic values inside the HTML elements.

`${rowHeaders}` – Row headers of the selected value cell.

`${columnHeaders}` – Column headers of the selected value cell.

`${rowFields}` – Row fields of the selected value cell.

`${columnFields}` – Column fields of the selected value cell.

`${valueField}` – Field name of the selected value cell.

`${aggregateType}` – Aggregate type of the selected value cell.

`${value}` - Formatted value of the selected value cell.

The tooltip customization is common for both pivot table and pivot chart or it can be done individually as well. To customize the pivot table tooltip, the above procedure needs to be followed. To customize the pivot chart tooltip alone use `template` property of tooltip under [chartSettings](#).

In the below sample, the pivot table and pivot chart shows customized tooltip layouts.

INDEX.JSX

```

{% raw %}
import { Inject, PivotViewComponent, Toolbar } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let toolbarOptions = ['Grid', 'Chart'];
    let chartSettings = {
        chartSeries: { type: 'Column', animation: { enable: false } },
        tooltip: { template: '<span class="wrap">${aggregateType} of
${valueField}: ${value}</span>' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        filters: []
    };
    let pivotObj;

```

```

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} showTooltip={true} dataSourceSettings={dataSourceSettings}
chartSettings={chartSettings} displayOption={{ view: 'Both' }}
showToolbar={true} tooltipTemplate={"#Template"}
toolbar={toolbarOptions}><Inject
services={[Toolbar]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { IDataOptions, IDataset, Inject, PivotViewComponent, Toolbar } from
'@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
function App() {
    let toolbarOptions: any = ['Grid', 'Chart'];
    let chartSettings: ChartSettings = {
        chartSeries: { type: 'Column', animation: { enable: false } },
        tooltip: { template: '<span class="wrap">${aggregateType} of
${valueField}: ${value}</span>' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        filters: []
    }
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} showTooltip={true} dataSourceSettings={dataSourceSettings}
chartSettings={chartSettings} displayOption={{ view: 'Both' }}
showToolbar={true} tooltipTemplate={"#Template"}
toolbar={toolbarOptions}><Inject services={[Toolbar]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

Css customization in React Pivotview component

Hiding Axis

The visibility of row, column, value and filter axis in Field List and Grouping Bar can be changed using custom CSS setting. To do so, please refer the code sample below:

INDEX.JSX

```
import { CalculatedField, FieldList, Inject, PivotViewComponent, GroupingBar } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
const SAMPLE_CSS = `
/* Hiding column axis in grouping bar */
#PivotView .e-group-columns {
    display: none;
}
/* Increasing filter axis height to fill column axis portion */
#PivotView .e-group-filters {
    min-height: 74.67px !important;
}
/* Hiding column axis in field list */
.e-pivotfieldlist-wrapper .e-field-list-columns {
    display: none;
}
/* Increasing value axis height to fill column axis portion */
.e-pivotfieldlist-wrapper .e-field-list-values {
    margin-top: 0px !important;
    min-height: 338px !important;
}
.e-pivotfieldlist-wrapper .e-values {
    height: 310px !important;
}
/* Hiding row axis in grouping bar */
// #PivotView .e-group-rows {
//     display: none;
// }
/* Hiding row axis in field list */
// .e-pivotfieldlist-wrapper .e-field-list-rows {
//     display: none;
// }
/* Hiding value axis in grouping bar */
// #PivotView .e-group-values {
//     display: none;
// }
/* Hiding value axis in field list */
// .e-pivotfieldlist-wrapper .e-field-list-values {
//     display: none;
// }
/* Hiding filter axis in grouping bar */
// #PivotView .e-group-filters {
//     display: none;
// }
/* Hiding filter axis in field list */
// .e-pivotfieldlist-wrapper .e-field-list-filters {
//     display: none;
// }

```

```

    //}`;
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<div>
    <style>{SAMPLE_CSS}</style>
    <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}
showGroupingBar={true}><Inject services={[CalculatedField, FieldList,
GroupingBar]}></PivotViewComponent>
    </div>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { CalculatedField, FieldList, IDataOptions, IDataset, Inject,
PivotViewComponent, GroupingBar } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
const SAMPLE_CSS = `
/* Hiding column axis in grouping bar */
#PivotView .e-group-columns {
  display: none;
}
/* Increasing filter axis height to fill column axis portion */
#PivotView .e-group-filters {
  min-height: 74.67px !important;
}
/* Hiding column axis in field list */
.e-pivotfieldlist-wrapper .e-field-list-columns {
  display: none;
}
/* Increasing value axis height to fill column axis portion */
.e-pivotfieldlist-wrapper .e-field-list-values {
  margin-top: 0px !important;
  min-height: 338px !important;
}
.e-pivotfieldlist-wrapper .e-values {
  height: 310px !important;
}

```

```

/* Hiding row axis in grouping bar */
// #PivotView .e-group-rows {
//     display: none;
// }
/* Hiding row axis in field list */
// .e-pivotfieldlist-wrapper .e-field-list-rows {
//     display: none;
// }
/* Hiding value axis in grouping bar */
// #PivotView .e-group-values {
//     display: none;
// }
/* Hiding value axis in field list */
// .e-pivotfieldlist-wrapper .e-field-list-values {
//     display: none;
// }
/* Hiding filter axis in grouping bar */
// #PivotView .e-group-filters {
//     display: none;
// }
/* Hiding filter axis in field list */
// .e-pivotfieldlist-wrapper .e-field-list-filters {
//     display: none;
// }`;

function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (
        <div>
            <style>{SAMPLE_CSS}</style>
            <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}
showGroupingBar={true}><Inject services={[CalculatedField, FieldList,
GroupingBar]}></PivotViewComponent>
        </div>
    );
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Text Alignment

The alignment of text inside row headers, column headers, value cells and summary cells can be changed using custom CSS setting. To do so, please refer the code sample below:

INDEX.JSX

```
import { CalculatedField, FieldList, Inject, PivotViewComponent, GroupingBar } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
const SAMPLE_CSS = `
//Value Cells
.e-pivotview .e-valuescontent {
    text-align: center !important;
}
//Columns Headers
/* .e-pivotview .e-columnsheader {
    text-align: center !important;
}
//Rows Headers
.e-pivotview .e-rowsheader {
    text-align: center !important;
}*/
//Summary Cells
/* .e-pivotview .e-summary {
    text-align: center !important;
}*/`;
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (
        <div>
            <style>{SAMPLE_CSS}</style>
            <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
            height={350} dataSourceSettings={dataSourceSettings}
            allowCalculatedField={true} showFieldList={true}
            showGroupingBar={true}><Inject services={[CalculatedField, FieldList, GroupingBar]}> </PivotViewComponent>
        </div>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```

import { CalculatedField, FieldList, IDataOptions, IDataset, Inject,
PivotViewComponent, GroupingBar } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
const SAMPLE_CSS = `
//Value Cells
.e-pivotview .e-valuescontent {
    text-align: center !important;
}
//Columns Headers
/* .e-pivotview .e-columnsheader {
    text-align: center !important;
}
//Rows Headers
.e-pivotview .e-rowsheader {
    text-align: center !important;
}*/
//Summary Cells
/* .e-pivotview .e-summary {
    text-align: center !important;
}*/`;
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (
        <div>
            <style>{SAMPLE_CSS}</style>
            <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}
showGroupingBar={true}><Inject services={[CalculatedField, FieldList,
GroupingBar]}></PivotViewComponent>
        </div>
    );
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Customize header, value and summary cell style

The elements in pivot table like header cell, value cell and summary cell style can be customized using built-in CSS names. To do so, please refer the code sample below:

INDEX.JSX


```

import { CalculatedField, FieldList, Inject, PivotViewComponent, GroupingBar
} from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
const SAMPLE_CSS = `
  //Value Cells
  .e-pivotview .e-summary:not(.e-gtot) {
    background-color: pink !important;
  }
  //Columns Headers
  /*.e-pivotview .e-headercell {
    background-color: thistle !important;
  }
  //Rows Headers
  .e-pivotview .e-rowsheader {
    background-color: skyblue !important;
  }*/
  //Summary Cells
  /*.e-pivotview .e-gtot {
    background-color: greenYellow !important;
  }*/`;
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<div>
    <style>{SAMPLE_CSS}</style>
    <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}
showGroupingBar={true}><Inject services={[CalculatedField, FieldList,
GroupingBar]}></PivotViewComponent>
    </div>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { CalculatedField, FieldList, IDataOptions, IDataset, Inject,
PivotViewComponent, GroupingBar } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
const SAMPLE_CSS = `

```

```

//Value Cells
.e-pivotview .e-summary:not(.e-gtot) {
    background-color: pink !important;
}
//Columns Headers
/* .e-pivotview .e-headercell {
    background-color: thistle !important;
}
//Rows Headers
.e-pivotview .e-rowsheader {
    background-color: skyblue !important;
}*/
//Summary Cells
/* .e-pivotview .e-gtot {
    background-color: greenYellow !important;
}*/;
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (
        <div>
            <style>{SAMPLE_CSS}</style>
            <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowCalculatedField={true} showFieldList={true}
showGroupingBar={true}><Inject services={[CalculatedField, FieldList,
GroupingBar]}> </PivotViewComponent>
        </div>
    )
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Printing and Exporting

Print in React Pivotview component

The rendered pivot table can be printed directly from the browser by invoking the [print](#) method from the grid's instance. The below sample code illustrates the print option being invoked by an external button click.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```

import { pivotData } from './datasource';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
function App() {
  let dataSourceSettings = {
    dataSource: pivotData,
    expandAll: false,
    enableSorting: true,
    drilledMembers: [{ name: 'Country', items: ['France'] }],
    columns: [
      { name: 'Year', caption: 'Production Year' },
      { name: 'Quarter' },
    ],
    values: [
      { name: 'Sold', caption: 'Units Sold' },
      { name: 'Amount', caption: 'Sold Amount' },
    ],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  }
  let pivotObj;
  function btnClick() {
    pivotObj.grid.print();
  }
  return (<div>
    <div className='col-lg-3 property-section'>
      <ButtonComponent cssClass='e-primary' isPrimary={true}
onClick={btnClick.bind(this)}>Print</ButtonComponent>
    </div>
    <div className="col-md-9">
      <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>
    </div>
  </div>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: false,
    enableSorting: true,
    drilledMembers: [{ name: 'Country', items: ['France'] }],
    columns: [
      { name: 'Year', caption: 'Production Year' },
      { name: 'Quarter' },
    ],
  },

```

```

    values: [
      { name: 'Sold', caption: 'Units Sold' },
      { name: 'Amount', caption: 'Sold Amount' },
    ],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  }
  let pivotObj: PivotViewComponent;
  function btnClick(): void {
    pivotObj.grid.print();
  }
  return (<div>
    <div className='col-lg-3 property-section'>
      <ButtonComponent cssClass='e-primary' isPrimary={true}
onClick={btnClick.bind(this)}>Print</ButtonComponent>
    </div>
    <div className="col-md-9">
      <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>
    </div>
  </div>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Similarly, to print the pivot chart, use the [print](#) method from the chart's instance. The below sample code illustrates the print option being invoked by an external button click.

To use pivot chart, you need to inject the `PivotChart` module in the pivot table.

To display the pivot chart, set the [displayOption](#) property to either **Chart** or **Both**.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
function App() {
  let dataSourceSettings = {
    dataSource: pivotData,
    expandAll: false,
    enableSorting: true,
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    columns: [
      { name: 'Year', caption: 'Production Year' },
      { name: 'Quarter' },
    ],
    values: [
      { name: 'Sold', caption: 'Units Sold' },
      { name: 'Amount', caption: 'Sold Amount' },
    ],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  }

```

```

    }
    let pivotObj;
    function btnClick() {
        pivotObj.chart.print();
    }
    return (<div>
        <div className='col-lg-3 property-section'>
            <ButtonComponent cssClass='e-primary' isPrimary={true}
onClick={btnClick.bind(this)}>Print</ButtonComponent>
        </div>
        <div className="col-md-9">
            <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350}
            displayOption={{ view: 'Chart' }}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>
        </div>
    </div>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataset[],
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [
            { name: 'Year', caption: 'Production Year' },
            { name: 'Quarter' },
        ],
        values: [
            { name: 'Sold', caption: 'Units Sold' },
            { name: 'Amount', caption: 'Sold Amount' },
        ],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    let pivotObj: PivotViewComponent;
    function btnClick(): void {
        pivotObj.chart.print();
    }
    return (<div>
        <div className='col-lg-3 property-section'>
            <ButtonComponent cssClass='e-primary' isPrimary={true}
onClick={btnClick.bind(this)}>Print</ButtonComponent>
        </div>
    </div>);
}

```

```

    <div className="col-md-9">
      <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350}
      displayOption={{ view: 'Chart' }}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>
    </div>
  </div>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Excel export in React Pivot Table component

The Excel export allows Pivot Table data to be exported as Excel document. To enable Excel export in the pivot table, set the [allowExcelExport](#) property in [PivotView](#) to **true**. Once the API is set, user needs to call the [excelExport](#) method for exporting on external button click.

The pivot table component can be exported to Excel format using options available in the toolbar. For more details [refer](#) here.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function btnClick() {
    pivotObj.excelExport();
  }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
        <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        pivotObj.excelExport();
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Multiple pivot table exporting

The Excel export provides an option to export multiple pivot table data in the same Excel file.

Same WorkSheet

The Excel export provides support to export multiple pivot tables in same sheet. To export in same sheet, define `multipleExport.type` as `AppendToSheet` in `excelExportProperties`. It has an option to provide blank rows between pivot tables and these blank row(s) count can be defined using the `multipleExport.blankRows` property.

By default, `multipleExport.blankRows` value is 5 between pivot tables within the same sheet.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],

```

```

        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]}
    };
    let pivotObj;
    let dataSourceSettings1 = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]}
    };
    let pivotObj1;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
        <div className="col-md-9"> <PivotViewComponent ref={d => pivotObj1 = d}
id='PivotView1' height={350} dataSourceSettings={dataSourceSettings1}
allowExcelExport={true}></PivotViewComponent></div>
        <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let excelExportProperties = {
            multipleExport: { type: 'AppendToSheet', blankRows: 2 }
        };
        let firstGridExport =
pivotObj.grid.excelExport(excelExportProperties, true);
        firstGridExport.then((fData) => {
            pivotObj1.excelExport(excelExportProperties, false, fData);
        });
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
function App() {

```



```

let dataSourceSettings: IDataOptions = {
  columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
  dataSource: pivotData as IDataset[],
  expandAll: false,
  filters: [],
  formatSettings: [{ name: 'Amount', format: 'C0' }],
  rows: [{ name: 'Country' }, { name: 'Products' }],
  values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
}
let pivotObj: PivotViewComponent;
let dataSourceSettings1: IDataOptions = {
  columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
  dataSource: pivotData as IDataset[],
  expandAll: true,
  filters: [],
  formatSettings: [{ name: 'Amount', format: 'C0' }],
  rows: [{ name: 'Country' }, { name: 'Products' }],
  values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
}
let pivotObj1: PivotViewComponent;

return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
  <div className="col-md-9"> <PivotViewComponent ref={d => pivotObj1 = d}
id='PivotView1' height={350} dataSourceSettings={dataSourceSettings1}
allowExcelExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
function btnClick(): void {
  let excelExportProperties: ExcelExportProperties = {
    multipleExport: { type: 'AppendToSheet', blankRows: 2 }
  };
  let firstGridExport: Promise<any> =
pivotObj.grid.excelExport(excelExportProperties, true);
  firstGridExport.then((fData: any) => {
    pivotObj1.excelExport(excelExportProperties, false, fData);
  });
}
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

New Worksheet

Excel export provides support to export multiple pivot tables into new sheets. To export in new sheets, define `multipleExport.type` as `NewSheet` in `excelExportProperties`.

INDEX.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
```

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    let dataSourceSettings1 = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj1;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
    <div className="col-md-9"> <PivotViewComponent ref={d => pivotObj1 = d}
id='PivotView1' height={350} dataSourceSettings={dataSourceSettings1}
allowExcelExport={true}></PivotViewComponent></div>
    <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let excelExportProperties = {
            multipleExport: { type: 'NewSheet' }
        };
        let firstGridExport =
pivotObj.grid.excelExport(excelExportProperties, true);
        firstGridExport.then((fData) => {
            pivotObj1.excelExport(excelExportProperties, false, fData);
        });
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    let dataSourceSettings1: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj1: PivotViewComponent;

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
        <div className="col-md-9"> <PivotViewComponent ref={d => pivotObj1 = d}
id='PivotView1' height={350} dataSourceSettings={dataSourceSettings1}
allowExcelExport={true}></PivotViewComponent></div>
        <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        let excelExportProperties: ExcelExportProperties = {
            multipleExport: { type: 'NewSheet' }
        };
        let firstGridExport: Promise<any> =
pivotObj.grid.excelExport(excelExportProperties, true);
        firstGridExport.then((fData: any) => {
            pivotObj1.excelExport(excelExportProperties, false, fData);
        });
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Changing the pivot table style while exporting

The Excel export provides an option to change colors for headers, caption and records in pivot table before exporting. In-order to apply colors, define **theme** settings in **excelExportProperties** object and pass it as a parameter to the [excelExport](#) method.

By default, material theme will be applied to the pivot table during Excel exporting.

INDEX.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let excelExportProperties = {
            theme: {
                header: { fontName: 'Segoe UI', fontColor: '#666666' },
                record: { fontName: 'Segoe UI', fontColor: '#666666' },
                caption: { fontName: 'Segoe UI', fontColor: '#666666' }
            }
        };
        pivotObj.grid.excelExport(excelExportProperties);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
```

```

import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function btnClick(): void {
    let excelExportProperties: ExcelExportProperties = {
      theme:
      {
        header: { fontName: 'Segoe UI', fontColor: '#666666' },
        record: { fontName: 'Segoe UI', fontColor: '#666666' },
        caption: { fontName: 'Segoe UI', fontColor: '#666666' }
      }
    };
    pivotObj.grid.excelExport(excelExportProperties);
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Add header and footer while exporting

The Excel export provides an option to include header and footer content for the excel document before exporting. In-order to add header and footer, define **header** and **footer** properties in **excelExportProperties** object and pass it as a parameter to the [excelExport](#) method.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,

```

```

        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let excelExportProperties = {
            header: {
                headerRows: 2,
                rows: [
                    { cells: [{ colSpan: 4, value: "Pivot Table", style: {
fontColor: '#C67878', fontSize: 20, hAlign: 'Center', bold: true, underline:
true } }] }
                ]
            },
            footer: {
                footerRows: 4,
                rows: [
                    { cells: [{ colSpan: 4, value: "Thank you for your
business!", style: { hAlign: 'Center', bold: true } }] },
                    { cells: [{ colSpan: 4, value: "!Visit Again!", style: {
hAlign: 'Center', bold: true } }] }
                ]
            }
        };
        pivotObj.grid.excelExport(excelExportProperties);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
    };

```

```

    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
    <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);

  function btnClick(): void {
    let excelExportProperties: ExcelExportProperties = {
      header: {
        headerRows: 2,
        rows: [
          { cells: [{ colSpan: 4, value: "Pivot Table", style: {
fontColor: '#C67878', fontSize: 20, hAlign: 'Center', bold: true, underline:
true } }] }
        ],
      },
      footer: {
        footerRows: 4,
        rows: [
          { cells: [{ colSpan: 4, value: "Thank you for your business!",
style: { hAlign: 'Center', bold: true } }] },
          { cells: [{ colSpan: 4, value: "!Visit Again!", style: {
hAlign: 'Center', bold: true } }] }
        ]
      }
    };
    pivotObj.grid.excelExport(excelExportProperties);
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Changing the file name while exporting

The Excel export provides an option to change file name of the document before exporting. In-order to change the file name, define **fileName** property in **excelExportProperties** object and pass it as a parameter to the [excelExport](#) method.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {

```

```

    let dataSourceSettings = {
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      dataSource: pivotData,
      expandAll: false,
      filters: [],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
      let excelExportProperties = {
        fileName: 'sample.xlsx'
      };
      pivotObj.grid.excelExport(excelExportProperties);
    }
  }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}

```



```

dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
  <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>;
  function btnClick(): void {
    let excelExportProperties: ExcelExportProperties = {
      fileName: 'sample.xlsx'
    };
    pivotObj.grid.excelExport(excelExportProperties);
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Limitation when exporting millions of records to Excel format

By default, Microsoft Excel supports only 1,048,576 records in an Excel sheet. Hence, it is not possible to export millions of records to Excel. You can refer to the [documentation link](#) for more details on Microsoft Excel specifications and limits. Therefore, it is suggested to export the data in CSV (Comma-Separated Values) or other formats that can handle large datasets more efficiently than Excel.

CSV Export

The Excel export allows pivot table data to be exported in **CSV** file format as well. To enable CSV export in the pivot table, set the [allowExcelExport](#) property in [PivotView](#) as **true**. Once the API is set, user needs to call the [csvExport](#) method for exporting on external button click.

The pivot table component can be exported to CSV format using options available in the toolbar. For more details [refer](#) here.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>

```

```

    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>;
    function btnClick() {
        pivotObj.csvExport();
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>;
    function btnClick(): void {
        pivotObj.csvExport();
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Virtual Scroll Data

You can export the pivot table virtual scroll data as Excel/CSV document by using PivotEngine export without any performance degradation. To enable PivotEngine export in the pivot table, set the [allowExcelExport](#) as true. You need to use the `exportToExcel` method for PivotEngine export.

To use PivotEngine export, You need to inject the `ExcelExport` module in pivot table.

PivotEngine export will be performed while enabling virtual scrolling by default.

Virtual Scroll Data Excel Export

INDEX.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, Inject, ExcelExport } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} allowExcelExport={true}><Inject services={[ExcelExport]}></PivotViewComponent></div>
        <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-primary'
        onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        pivotObj.excelExportModule.exportToExcel('Excel');
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, Inject, ExcelExport } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} allowExcelExport={true}><Inject services={[ExcelExport]}></PivotViewComponent></div>
        <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-primary'
        onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        pivotObj.excelExportModule.exportToExcel('Excel');
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

```

    }
    let pivotObj: PivotViewComponent;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings} allowExcelExport={true}><Inject
services={[ExcelExport]} /></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        pivotObj.excelExportModule.exportToExcel('Excel');
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Virtual Scroll Data CSV Export

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, Inject, ExcelExport } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings} allowExcelExport={true}><Inject
services={[ExcelExport]} /></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let excelExportProperties = {
            fileName: 'csvexport.csv',
        };
        pivotObj.csvExport(excelExportProperties);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, Inject, ExcelExport }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings} allowExcelExport={true}><Inject
services={[ExcelExport]} /></PivotViewComponent></div>
    <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function btnClick(): void {
    let excelExportProperties = {
      fileName: 'csvexport.csv',
    };
    pivotObj.csvExport(excelExportProperties);
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

[Export all pages](#)

The pivot engine exports the entire virtual data of the pivot table (i.e. the data that contains all of the records used to render the complete pivot table) as an Excel/CSV document. To export just the current viewport of the pivot table, set the [exportAllPages](#) property to **false**. To use the pivot engine export, add the [ExcelExport](#) module into the pivot table.

By default, the pivot engine export will be performed while virtual scrolling is enabled.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, VirtualScroll, Inject } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {

```

```

    let dataSourceSettings = {
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      dataSource: pivotData,
      expandAll: true,
      filters: [],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      values: [{ name: 'Country' }, { name: 'Products' }],
      rows: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} exportAllPages='false'
allowExcelExport={true} enableVirtualization={true}
dataSourceSettings={dataSourceSettings}><Inject
services={[VirtualScroll]}/></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
      pivotObj.excelExport();
    }
  }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, VirtualScroll, Inject }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: true,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    values: [{ name: 'Country' }, { name: 'Products' }],
    rows: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} exportAllPages='false'
allowExcelExport={true} enableVirtualization={true}
dataSourceSettings={dataSourceSettings}><Inject
services={[VirtualScroll]}/></PivotViewComponent></div>

```

```

    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>;
    function btnClick(): void {
        pivotObj.excelExport();
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Events

ExcelQueryCellInfo

The event `excelQueryCellInfo` triggers while framing each row and value cell during Excel export. It allows the user to customize the cell value, style etc. of the current cell. It has the following parameters:

- `value` - It holds the cell value.
- `column` - It holds column information for the current cell.
- `data` - It holds the entire row data across the current cell.
- `style` - It holds the style properties for the cell.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        columnWidth: 140,
        excelQueryCellInfo: excelQueryCellInfo.bind(this)
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    function excelQueryCellInfo(args) {
        //triggers every time for header cell while rendering
    }
    return <div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>

```

```

    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary' onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>;
    function btnClick() {
        pivotObj.excelExport();
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
function App() {
    let gridSettings: GridSettings = {
        columnWidth: 140,
        excelQueryCellInfo: excelQueryCellInfo.bind(this)
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    function excelQueryCellInfo(args: ExcelQueryCellInfoEventArgs): void {
        //triggers every time for header cell while rendering
    }

    return <div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary' onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>

    function btnClick(): void {
        pivotObj.excelExport();
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```


[ExcelHeaderQueryCellInfo](#)

The event `excelHeaderQueryCellInfo` triggers on framing each header cell during Excel export. It allows the user to customize the cell value, style etc. of the current cell. It has the following parameters:

- `cell` - It holds the current cell information.
- `style` - It holds the style properties for the cell.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        columnWidth: 140,
        excelHeaderQueryCellInfo: excelHeaderQueryCellInfo.bind(this)
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    function excelHeaderQueryCellInfo(args) {
        //triggers every time for header cell while rendering
    }
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        pivotObj.excelExport();
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
```

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
  let gridSettings: GridSettings = {
    columnWidth: 140,
    excelHeaderQueryCellInfo: excelHeaderQueryCellInfo.bind(this)
  } as GridSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  function excelHeaderQueryCellInfo(args:
ExcelHeaderQueryCellInfoEventArgs): void {
    //triggers every time for header cell while rendering
  }

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}
allowExcelExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);

  function btnClick(): void {
    pivotObj.excelExport();
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

ExportComplete

The event [exportComplete](#) is triggered after the pivot table data has been exported to a Excel/CSV document. You can use this event to acquire blob stream data for further customization and processing at your end by passing the `isBlob` parameter as **true** when using the [excelExport](#) method. It has the following parameters:

- **type** - It holds the current export type such as PDF, Excel, and CSV.
- **promise** - It holds the promise object for blob data.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, VirtualScroll, Inject, ExcelExport } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        values: [{ name: 'Country' }, { name: 'Products' }],
        rows: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowExcelExport={true}
enableVirtualization={true} dataSourceSettings={dataSourceSettings}
exportComplete={exportComplete.bind(this)}><Inject services={[VirtualScroll,
ExcelExport]}></PivotViewComponent></div>
        <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function exportComplete(args) {
        args.promise.then((e) => {
            console.log(e.blobData);
        });
    }
    function btnClick() {
        let excelExportProperties = {
            fileName: 'excelexport.xlsx'
        };
        pivotObj.excelExport(excelExportProperties, false, null, true);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, VirtualScroll, Inject,
ExportCompleteEventArgs, ExcelExport } from '@syncfusion/ej2-react-
pivotview';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {

```

```

    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: true,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    values: [{ name: 'Country' }, { name: 'Products' }],
    rows: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowExcelExport={true}
enableVirtualization={true} dataSourceSettings={dataSourceSettings}
exportComplete={exportComplete.bind(this)}><Inject services={[VirtualScroll,
ExcelExport]}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function exportComplete(args: ExportCompleteEventArgs): void {
    args.promise.then((e: { blobData: Blob }) => {
      console.log(e.blobData);
    });
  }
  function btnClick(): void {
    let excelExportProperties: ExcelExportProperties = {
      fileName: 'excelexport.xlsx'
    };
    pivotObj.excelExport(excelExportProperties, false, null, true);
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

[See Also](#)

- [PDF Exporting](#)

Pdf export in React Pivotview component

PDF export allows exporting pivot table data as PDF document. To enable PDF export in the pivot table, set the `allowPdfExport` as true. You need to use the `pdfExport` method for PDF exporting.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,

```

```

        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        pivotObj.pdfExport();
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        pivotObj.pdfExport();
    }
}
;

```

```
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Multiple pivot table exporting

PDF export provides an option for exporting multiple pivot tables to same file. In this exported document, each pivot table will be exported to new page of document in same file.

INDEX.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    let dataSourceSettings1 = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj1;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
    <div className="col-md-9"> <PivotViewComponent ref={d => pivotObj1 = d}
id='PivotView1' height={280} dataSourceSettings={dataSourceSettings1}
allowPdfExport={true}></PivotViewComponent></div>
    <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let firstGridPdfExport = pivotObj.grid.pdfExport({}, true);
        firstGridPdfExport.then((pdfData) => {
            pivotObj1.pdfExport({}, false, pdfData);
        });
    }
}
```

```
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    let dataSourceSettings1: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj1: PivotViewComponent;

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
        <div className="col-md-9"> <PivotViewComponent ref={d => pivotObj1 = d}
id='PivotView1' height={280} dataSourceSettings={dataSourceSettings1}
allowPdfExport={true}></PivotViewComponent></div>
        <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        let firstGridPdfExport: Promise<Object> = pivotObj.grid.pdfExport({},
true);
        firstGridPdfExport.then((pdfData: Object) => {
            pivotObj1.pdfExport({}, false, pdfData);
        });
    }
};
```

```
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Export table and chart into the same document

When the [displayOption](#) is set to **Both**, you can export both the table and the chart into the same PDF document. To achieve this, use the [pdfExport](#) method and set the [exportBothTableAndChart](#) parameter to **true**.

INDEX.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, VirtualScroll, Inject, PDFExport, PivotChart }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let chartSettings = {
        chartSeries: { type: 'Column' }
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
enableVirtualization={true}
        dataSourceSettings={dataSourceSettings} displayOption={{ view:
'Both' }} chartSettings={chartSettings}>
            <Inject services={[VirtualScroll, PivotChart,
PDFExport]} /></PivotViewComponent></div>
            <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let pdfExportProperties = {
            fileName: 'pdfexport'
        };
        pivotObj.pdfExport(pdfExportProperties, false, null, false, true);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX


```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, VirtualScroll, Inject,
PDFExport, PivotChart } from '@syncfusion/ej2-react-pivotview';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartSettings';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
    let chartSettings: ChartSettings = {
        chartSeries: { type: 'Column' }
    } as ChartSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
enableVirtualization={true}
        dataSourceSettings={dataSourceSettings} displayOption={{ view: 'Both' }}
        chartSettings={chartSettings}>
            <Inject services={[VirtualScroll, PivotChart,
PDFExport]}></PivotViewComponent></div>
            <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        let pdfExportProperties: PdfExportProperties = {
            fileName: 'pdfexport'
        };
        pivotObj.pdfExport(pdfExportProperties, false, null, false, true);
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Customization during PDF export

PDF export provides option to customize mapping of pivot table to the exported PDF document.

To add header and footer while exporting

You can customize text, page number, line, page size and changing orientation in header and footer of the exported document.

To add a text in header/footer

You can add text either in header or footer of the exported PDF document like in the below code example.

```
`ts
```

```
let pdfExportProperties: PdfExportProperties = {  
  header: {  
    fromTop: 0,  
    height: 130,  
    contents: [  
      {  
        type: 'Text',  
        value: "Northwind Traders",  
        position: { x: 0, y: 50 },  
        style: { textBrushColor: '#000000', fontSize: 13 }  
      },  
    ]  
  }  
},  
,
```

```
`ts
```

```
let pdfExportProperties = {  
  header: {  
    fromTop: 0,  
    height: 130,  
    contents: [  
      {  
        type: 'Text',  
        value: "Northwind Traders",  
        position: { x: 0, y: 50 },  
        style: { textBrushColor: '#000000', fontSize: 13 }  
      },  
    ]  
  }  
};
```

`

To draw a line in header/footer

You can add line either in header or footer of the exported PDF document like in the below code example.

Supported line styles:

- dash
- dot
- dashdot
- dashdotdot
- solid

`ts

```
let pdfExportProperties: PdfExportProperties = {  
  header: {  
    fromTop: 0,  
    height: 130,  
    contents: [  
      {  
        type: 'Line',  
        style: { penColor: '#000080', penSize: 2, dashStyle: 'Solid' },  
        points: { x1: 0, y1: 4, x2: 685, y2: 4 }  
      }  
    ]  
  }  
}
```

`

`ts

```
let pdfExportProperties = {  
  header: {  
    fromTop: 0,  
    height: 130,  
    contents: [  
      {  
        type: 'Line',  
        style: { penColor: '#000080', penSize: 2, dashStyle: 'Solid' },
```

```
points: { x1: 0, y1: 4, x2: 685, y2: 4 }
}
]
}
};
`ts
```

[Add page number in header/footer](#)

You can add page number either in header or footer of exported PDF document like in the below code example.

Supported page number types:

- LowerLatin - a, b, c,
- UpperLatin - A, B, C,
- LowerRoman - i, ii, iii,
- UpperRoman - I, II, III,
- Number - 1,2,3.

```
`ts
let pdfExportProperties: PdfExportProperties = {
  header: {
    fromTop: 0,
    height: 130,
    contents: [
      {
        type: 'PageNumber',
        pageNumberType: 'Arabic',
        format: 'Page {$current} of {$total}', //optional
        position: { x: 0, y: 25 },
        style: { textBrushColor: '#ffff80', fontSize: 15, hAlign: 'Center' }
      }
    ]
  }
}
`ts

let pdfExportProperties = {
```

```

header: {
  fromTop: 0,
  height: 130,
  contents: [
    {
      type: 'PageNumber',
      pageNumberType: 'Arabic',
      format: 'Page {$current} of {$total}',
      position: { x: 0, y: 25 },
      style: { textBrushColor: '#ffff80', fontSize: 15, hAlign: 'Center' }
    }
  ]
};

```

The below code illustrates the PDF export customization options.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
<div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function btnClick() {
    let pdfExportProperties = {

```

```

        header: {
            fromTop: 0,
            height: 130,
            contents: [
                {
                    type: 'Text',
                    value: "Pivot Table",
                    position: { x: 0, y: 50 },
                    style: { textBrushColor: '#000000', fontSize: 13,
dashStyle: 'Solid', hAlign: 'Center' }
                }
            ]
        },
        footer: {
            fromBottom: 160,
            height: 150,
            contents: [
                {
                    type: 'PageNumber',
                    pageNumberType: 'Arabic',
                    format: 'Page {$current} of {$total}',
                    position: { x: 0, y: 25 },
                    style: { textBrushColor: '#02007a', fontSize: 15 }
                }
            ]
        }
    };
    pivotObj.pdfExport(pdfExportProperties);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

```

```

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        let pdfExportProperties: PdfExportProperties = {
            header: {
                fromTop: 0,
                height: 130,
                contents: [
                    {
                        type: 'Text',
                        value: "Pivot Table",
                        position: { x: 0, y: 50 },
                        style: { textBrushColor: '#000000', fontSize: 13,
dashStyle:'Solid',hAlign:'Center' }
                    }
                ]
            },
            footer: {
                fromBottom: 160,
                height: 150,
                contents: [
                    {
                        type: 'PageNumber',
                        pageNumberType: 'Arabic',
                        format: 'Page { $current } of { $total }',
                        position: { x: 0, y: 25 },
                        style: { textBrushColor: '#02007a', fontSize: 15 }
                    }
                ]
            }
        };
        pivotObj.pdfExport(pdfExportProperties);
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Add an image in header/footer

You can add image (Base64 string) either in header or footer of the exported PDF document like in the below code example.

`ts

```

let pdfExportProperties = {
    header: {
        fromTop: 0,
        height: 130,

```

```

contents: [
{
type: 'Image',
src: image,
position: { x: 20, y: 10 },
size: { height: 100, width: 100 },
}
]
}
};
`

```

The below code illustrates the PDF export customization options.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let pdfExportProperties = {
            header: {
                fromTop: 0,
                height: 130,
                contents: [
                    {
                        type: 'Image',
                        src: image,

```



```

                position: { x: 20, y: 10 },
                size: { height: 100, width: 100 },
            }
        ]
    }
};
pivotObj.pdfExport(pdfExportProperties);
}
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import { image } from './image';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    };
    let pivotObj: PivotViewComponent;

    return (
        <div><div className="col-md-9"> <PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={280} dataSourceSettings={dataSourceSettings} allowPdfExport={true}></PivotViewComponent></div>
        <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-primary' onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>
    );
    function btnClick(): void {
        let pdfExportProperties: PdfExportProperties = {
            header: {
                fromTop: 0,
                height: 130,
                contents: [
                    {
                        type: 'Image',
                        src: image,
                        position: { x: 20, y: 10 },
                        size: { height: 100, width: 100 },
                    }
                ]
            }
        };
    }
}

```

```

    ]
  }
};
pivotObj.pdfExport(pdfExportProperties);
}
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Changing the file name while exporting

The PDF export provides an option to change file name of the document before exporting. In-order to change the file name, define **fileName** property in **pdfExportProperties** object and pass it as a parameter to the [pdfExport](#) method.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
<div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function btnClick() {
    let pdfExportProperties = {
      fileName: 'sample.pdf'
    };
    pivotObj.pdfExport(pdfExportProperties);
  }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';

```

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
<div className='col-lg-3 property-section'><ButtonComponent cssClass='e-primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        let pdfExportProperties: PdfExportProperties = {
            fileName: 'sample.pdf'
        };
        pivotObj.pdfExport(pdfExportProperties);
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Changing page orientation while exporting

The PDF export provides an option to change page orientation of the document before exporting. In order to change the page orientation, define **pageOrientation** property in **pdfExportProperties** object and pass it as a parameter to the [pdfExport](#) method. By default, the page orientation will be in **Portrait** and it can be changed to **Landscape** based on user requirement.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,

```

```

        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let pdfExportProperties = {
            pageOrientation: 'Landscape'
        };
        pivotObj.pdfExport(pdfExportProperties);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        let pdfExportProperties: PdfExportProperties = {

```

```

        pageOrientation: 'Landscape'
    };
    pivotObj.pdfExport(pdfExportProperties);
}
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Changing page size while exporting

The PDF export provides an option to change page size of the document before exporting. In-order to change the page size, define **pageSize** property in **pdfExportProperties** object and pass it as a parameter to the [pdfExport](#) method.

Supported page sizes are: Letter, Note, Legal, A0, A1, A2, A3, A5, A6, A7, A8, A9, B0, B1, B2, B3, B4, B5, Archa, Archb, Archc, Archd, Arche, Flsa, HalfLetter, Letter11x17, Ledger.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
    <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let pdfExportProperties = {
            pageSize: 'Letter'
        };
        pivotObj.pdfExport(pdfExportProperties);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
        <div className="col-lg-3 property-section"><ButtonComponent cssClass="e-
primary"
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        let pdfExportProperties: PdfExportProperties = {
            pageSize: 'Letter'
        };
        pivotObj.pdfExport(pdfExportProperties);
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Changing document width and height while exporting

Before exporting, you can change the height and width of the PDF document. To achieve this, use the **height** and **width** properties in the [beforeExport](#) event.

This option is only available if [enableVirtualization](#) is set to **true**. In addition, the **VirtualScroll** and **PDFExport** modules must be injected into the pivot table.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, VirtualScroll, Inject, PDFExport } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
    let dataSourceSettings = {

```

```

        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
enableVirtualization={true}
        dataSourceSettings={dataSourceSettings}
beforeExport={beforeExport.bind(this)}><Inject services={[VirtualScroll,
PDFExport]}></PivotViewComponent></div>
        <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function beforeExport(args) {
        args.width = pivotObj.element.offsetWidth;
        args.height = pivotObj.element.offsetHeight;
    }
    function btnClick() {
        let pdfExportProperties = {
            fileName: 'pdfexport'
        };
        pivotObj.pdfExport(pdfExportProperties, false, null, false, true);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, VirtualScroll, Inject,
BeforeExportEventArgs, PDFExport } from '@syncfusion/ej2-react-pivotview';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
}

```

```

let pivotObj: PivotViewComponent;
return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
enableVirtualization={true}
  dataSourceSettings={dataSourceSettings}
  beforeExport={beforeExport.bind(this)}><Inject services={[VirtualScroll,
PDFExport]} /></PivotViewComponent></div>
  <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
function beforeExport(args: BeforeExportEventArgs): void {
  args.width = pivotObj.element.offsetWidth;
  args.height = pivotObj.element.offsetHeight;
}
function btnClick(): void {
  let pdfExportProperties: PdfExportProperties = {
    fileName: 'pdfexport'
  };
  pivotObj.pdfExport(pdfExportProperties, false, null, false, true);
}
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Customize the table column count while exporting

Before exporting, you can split and export the pivot table columns on each page of the PDF document by using the **columnSize** property in the [beforeExport](#) event.

This option is only available if [enableVirtualization](#) is set to **true**. In addition, the **VirtualScroll** and **PDFExport** modules must be injected into the pivot table.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, VirtualScroll, Inject, PDFExport } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
enableVirtualization={true}

```



```

        dataSourceSettings={dataSourceSettings}
beforeExport={beforeExport.bind(this)}><Inject services={[VirtualScroll,
PDFExport]}/></PivotViewComponent></div>
        <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>;
        function beforeExport(args) {
            args.columnSize = 6;
        }
        function btnClick() {
            let pdfExportProperties = {
                fileName: 'pdfexport'
            };
            pivotObj.pdfExport(pdfExportProperties, false, null, false, true);
        }
    }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, VirtualScroll, Inject,
BeforeExportEventArgs, PDFExport } from '@syncfusion/ej2-react-pivotview';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
enableVirtualization={true}
        dataSourceSettings={dataSourceSettings}
beforeExport={beforeExport.bind(this)}><Inject services={[VirtualScroll,
PDFExport]}/></PivotViewComponent></div>
        <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>;
        function beforeExport(args: BeforeExportEventArgs): void {
            args.columnSize = 6;
        }
        function btnClick(): void {
            let pdfExportProperties: PdfExportProperties = {

```

```

        fileName: 'pdfexport'
    };
    pivotObj.pdfExport(pdfExportProperties, false, null, false, true);
}
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Changing the table's column width and row height while exporting

You can change the column width and row height in the PDF document during the pivot table export by using the [onPdfCellRender](#) event. Within this event, the `args.column.width` property allows you to change the width of specific columns.

As shown in the code example below, the “Unit Sold” column under “FY 2015” is changed to a width of 60 pixels.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, VirtualScroll, Inject, PDFExport } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' } ]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
enableVirtualization={true}
        dataSourceSettings={dataSourceSettings}
onPdfCellRender={onPdfCellRender.bind(this)}><Inject
services={[VirtualScroll, PDFExport]}></PivotViewComponent></div>
        <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function onPdfCellRender(args) {
        if (args.pivotCell && args.pivotCell.valueSort &&
args.pivotCell.valueSort.levelName === 'FY 2015.Units Sold') {
            args.column.width = 60
        }
    }
    function btnClick() {
        let pdfExportProperties = {
            fileName: 'pdfexport'
        };
    };

```

```

        pivotObj.pdfExport(pdfExportProperties, false, null, false, true);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, VirtualScroll, Inject,
PdfCellRenderArgs, PDFExport } from '@syncfusion/ej2-react-pivotview';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
enableVirtualization={true}
        dataSourceSettings={dataSourceSettings}
onPdfCellRender={onPdfCellRender.bind(this)}><Inject
services={[VirtualScroll, PDFExport]}></PivotViewComponent></div>
        <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function onPdfCellRender(args: PdfCellRenderArgs): void {
        if (args.pivotCell && args.pivotCell.valueSort &&
args.pivotCell.valueSort.levelName === 'FY 2015.Units Sold') {
            args.column.width = 60
        }
    }
    function btnClick(): void {
        let pdfExportProperties: PdfExportProperties = {
            fileName: 'pdfexport'
        };
        pivotObj.pdfExport(pdfExportProperties, false, null, false, true);
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Similarly, you can change the height of specific rows in the PDF document by using the `args.cell.height` property in the [onPdfCellRender](#) event.

As shown in the code example below, the “**Mountain Bikes**” row under “**France**” is changed to a height of **30** pixels.

INDEX.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, VirtualScroll, Inject, PDFExport } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
        'Quarter' }],
        dataSource: pivotData,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
        caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
    pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
    enableVirtualization={true}
        dataSourceSettings={dataSourceSettings}
    onPdfCellRender={onPdfCellRender.bind(this)}><Inject
    services={[VirtualScroll, PDFExport]}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
    primary'
    onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function onPdfCellRender(args) {
        if (args.pivotCell && args.pivotCell.valueSort &&
        args.pivotCell.valueSort.levelName === 'France.Mountain Bikes') {
            args.cell.height = 30
        }
    }
    function btnClick() {
        let pdfExportProperties = {
            fileName: 'pdfexport'
        };
        pivotObj.pdfExport(pdfExportProperties, false, null, false, true);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
```

```

import { IDataOptions, IDataset, PivotViewComponent, VirtualScroll, Inject,
PdfCellRenderArgs, PDFExport } from '@syncfusion/ej2-react-pivotview';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
enableVirtualization={true}
  dataSourceSettings={dataSourceSettings}
onPdfCellRender={onPdfCellRender.bind(this)}><Inject
services={[VirtualScroll, PDFExport]} /></PivotViewComponent></div>
  <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function onPdfCellRender(args: PdfCellRenderArgs): void {
    if (args.pivotCell && args.pivotCell.valueSort &&
args.pivotCell.valueSort.levelName === 'France.Mountain Bikes') {
      args.cell.height = 30
    }
  }
  function btnClick(): void {
    let pdfExportProperties: PdfExportProperties = {
      fileName: 'pdfexport'
    };
    pivotObj.pdfExport(pdfExportProperties, false, null, false, true);
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

This option is only available if [enableVirtualization](#) is set to **true**. In addition, the [VirtualScroll](#) and [PDFExport](#) modules must be injected into the pivot table.

Changing the pivot table style while exporting

The PDF export provides an option to change colors for headers, caption and records in pivot table before exporting. In-order to apply colors, define **theme** settings in **pdfExportProperties** object and pass it as a parameter to the [pdfExport](#) method.

By default, material theme will be applied to the pivot table during PDF exporting.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let pdfExportProperties = {
            theme: {
                header: {
                    fontColor: '#64FA50', fontName: 'Calibri', fontSize: 17,
bold: true, borders: { color: '#64FA50', lineStyle: 'Thin' }
                },
                record: {
                    fontColor: '#64FA50', fontName: 'Calibri', fontSize: 17,
bold: true
                },
                caption: {
                    fontColor: '#64FA50', fontName: 'Calibri', fontSize: 17,
bold: true
                }
            }
        };
        pivotObj.pdfExport(pdfExportProperties);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```

import { pivotData } from './datasource';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
  <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function btnClick(): void {
    let pdfExportProperties: PdfExportProperties = {
      theme: {
        header: {
          fontColor: '#64FA50', fontName: 'Calibri', fontSize: 17, bold:
true, borders: { color: '#64FA50', lineStyle: 'Thin' }
        },
        record: {
          fontColor: '#64FA50', fontName: 'Calibri', fontSize: 17, bold:
true
        },
        caption: {
          fontColor: '#64FA50', fontName: 'Calibri', fontSize: 17, bold:
true
        }
      }
    };
    pivotObj.pdfExport(pdfExportProperties);
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

<!-- markdownlint-disable MD009 -->

Changing default font while exporting

By default, the pivot table uses "Helvetica" font in the exported document. But it can be changed using the [theme](#) property in [pdfExportProperties](#).

The available built-in fonts are,

- Helvetica

- TimesRoman
- Courier
- Symbol
- ZapfDingbats

INDEX.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData, base64AlgeriaFont } from './datasource';
import { PdfTrueTypeFont } from '@syncfusion/ej2-pdf-export';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
<div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        let pdfExportProperties = {
            theme: {
                header: { font: new PdfTrueTypeFont(base64AlgeriaFont, 11)
            },
                caption: { font: new PdfTrueTypeFont(base64AlgeriaFont, 9)
            },
                record: { font: new PdfTrueTypeFont(base64AlgeriaFont, 10) }
            }
        };
        pivotObj.pdfExport(pdfExportProperties);
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
```



```

import * as ReactDOM from 'react-dom';
import { pivotData, base64AlgeriaFont } from './datasource';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import { PdfTrueTypeFont } from '@syncfusion/ej2-pdf-export';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function btnClick(): void {
    let pdfExportProperties: PdfExportProperties = {
      theme: {
        header: { font: new PdfTrueTypeFont(base64AlgeriaFont, 11)
},
        caption: { font: new PdfTrueTypeFont(base64AlgeriaFont, 9)
},
        record: { font: new PdfTrueTypeFont(base64AlgeriaFont, 10) }
      }
    };
    pivotObj.pdfExport(pdfExportProperties);
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Virtual Scroll Data

You can export the pivot table virtual scroll data as PDF document by using PivotEngine export without any performance degradation. To enable PivotEngine export in the pivot table, set the `allowPdfExport` as true. You need to use the `exportToPDF` method for PivotEngine export.

To use PivotEngine export, You need to inject the `PDFExport` module in pivot table.

PivotEngine export will be performed while enabling virtual scrolling by default

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, Inject, PDFExport } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```

import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings} allowPdfExport={true}><Inject
services={[PDFExport]} /></PivotViewComponent></div>
  <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function btnClick() {
    pivotObj.pdfExportModule.exportToPDF();
  }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, Inject, PDFExport }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings} allowPdfExport={true}><Inject
services={[PDFExport]} /></PivotViewComponent></div>

```

```

    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>;
    function btnClick(): void {
        pivotObj.pdfExportModule.exportToPDF();
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Repeat row headers

Repeat row headers on each page can be achieved using PivotEngine export option. To disable repeat row headers, you need to set `allowRepeatHeader` to `false` in `beforeExport` event. You need to use the `exportToPDF` method for PivotEngine export.

To use PivotEngine export, You need to inject the `PDFExport` module in pivot table.

By default, repeat row headers is enabled in the PivotEngine export.

INDEX.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, Inject, PDFExport } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' } ]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings} allowPdfExport={true}><Inject
services={[PDFExport]}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>;
        function btnClick() {
            pivotObj.pdfExportModule.exportToPDF();
        }
    }
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, Inject, PDFExport }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={280}
dataSourceSettings={dataSourceSettings} allowPdfExport={true}><Inject
services={[PDFExport]} /></PivotViewComponent></div>
  <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function btnClick(): void {
    pivotObj.pdfExportModule.exportToPDF();
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Export all pages

The pivot engine exports the entire virtual data of the pivot table (i.e. the data that contains all of the records used to render the complete pivot table) as a PDF document. To export just the current viewport of the pivot table, set the [exportAllPages](#) property to **false**. To use the pivot engine export, add the **PDFExport** module into the pivot table.

By default, the pivot engine export will be performed while virtual scrolling is enabled.

INDEX.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, VirtualScroll, Inject } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
```

```

        dataSource: pivotData,
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        values: [{ name: 'Country' }, { name: 'Products' }],
        rows: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} exportAllPages='false'
allowPdfExport={true} enableVirtualization={true}
dataSourceSettings={dataSourceSettings}><Inject
services={[VirtualScroll]}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        pivotObj.pdfExport();
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, VirtualScroll, Inject }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: true,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        values: [{ name: 'Country' }, { name: 'Products' }],
        rows: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} exportAllPages='false'
allowPdfExport={true} enableVirtualization={true}
dataSourceSettings={dataSourceSettings}><Inject
services={[VirtualScroll]}></PivotViewComponent></div>
    <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        pivotObj.pdfExport();
    }
}

```

```

    }
  };
  export default App;
  ReactDOM.render(<App />, document.getElementById('sample'));

```

Events

PdfQueryCellInfo

The event **pdfQueryCellInfo** triggers on framing each row and value cell during PDF export. It allows the user to customize the cell value, style etc. of the current cell. It has the following parameters:

- **value** - It holds the cell value.
- **column** - It holds column information for the current cell.
- **data** - It holds the entire row data across the current cell.
- **style** - It holds the style properties for the cell.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let gridSettings = {
    columnWidth: 140,
    pdfQueryCellInfo: pdfQueryCellInfo.bind(this)
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  function pdfQueryCellInfo(args) {
    //triggers every time for header cell while rendering
  }
  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
  <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function btnClick() {
    pivotObj.pdfExport();
  }
}

```

```

}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
    let gridSettings: GridSettings = {
        columnWidth: 140,
        pdfQueryCellInfo: pdfQueryCellInfo.bind(this)
    } as GridSettings;
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;
    function pdfQueryCellInfo(args: PdfQueryCellInfoEventArgs): void {
        //triggers every time for header cell while rendering
    }

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
        <div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick(): void {
        pivotObj.pdfExport();
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

PdfHeaderQueryCellInfo

The event `pdfHeaderQueryCellInfo` triggers on framing each column header cell during PDF export. It allows the user to customize the cell value, style etc. of the current cell. It has the following parameters:

- **cell** - It holds the current rendering cell information.
- **style** - It holds the style properties for the cell.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let gridSettings = {
        columnWidth: 140,
        pdfHeaderQueryCellInfo: pdfHeaderQueryCellInfo.bind(this)
    };
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    function pdfHeaderQueryCellInfo(args) {
        //triggers every time for header cell while rendering
    }
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
<div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
    function btnClick() {
        pivotObj.pdfExport();
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
```



```
function App() {
  let gridSettings: GridSettings = {
    columnWidth: 140,
    pdfHeaderQueryCellInfo: pdfHeaderQueryCellInfo.bind(this)
  } as GridSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  function pdfHeaderQueryCellInfo(args: PdfHeaderQueryCellInfoEventArgs):
void {
    //triggers every time for header cell while rendering
  }

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}
allowPdfExport={true}></PivotViewComponent></div>
    <div className="col-lg-3 property-section"><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function btnClick(): void {
    pivotObj.pdfExport();
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

ExportComplete

The event [exportComplete](#) is triggered after the pivot table data has been exported to a PDF document. You can use this event to acquire blob stream data for further customization and processing at your end by passing the `isBlob` parameter as `true` when using the [pdfExport](#) method. It has the following parameters:

- `type` - It holds the current export type such as PDF, Excel, and CSV.
- `promise` - It holds the promise object for blob data.

INDEX.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, VirtualScroll, Inject, PDFExport } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
```

```

function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: true,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    values: [{ name: 'Country' }, { name: 'Products' }],
    rows: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
enableVirtualization={true} dataSourceSettings={dataSourceSettings}
exportComplete={exportComplete.bind(this)}><Inject services={[VirtualScroll,
PDFExport]}></PivotViewComponent></div>
<div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function exportComplete(args) {
    args.promise.then((e) => {
      console.log(e.blobData);
    });
  }
  function btnClick() {
    let pdfExportProperties = {
      fileName: 'pdfexport.pdf'
    };
    pivotObj.pdfExport(pdfExportProperties, false, null, true);
  }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, VirtualScroll, Inject,
ExportCompleteEventArgs, PDFExport } from '@syncfusion/ej2-react-pivotview';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: true,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    values: [{ name: 'Country' }, { name: 'Products' }],

```

```

    rows: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350} allowPdfExport={true}
enableVirtualization={true} dataSourceSettings={dataSourceSettings}
exportComplete={exportComplete.bind(this)}><Inject services={[VirtualScroll,
PDFExport]}></PivotViewComponent></div><div className='col-lg-3 property-section'><ButtonComponent cssClass='e-
primary'
onClick={btnClick.bind(this)}>Export</ButtonComponent></div></div>);
  function exportComplete(args: ExportCompleteEventArgs): void {
    args.promise.then((e: { blobData: Blob }) => {
      console.log(e.blobData);
    });
  }
  function btnClick(): void {
    let pdfExportProperties: PdfExportProperties = {
      fileName: 'pdfexport.pdf'
    };
    pivotObj.pdfExport(pdfExportProperties, false, null, true);
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

[See Also](#)

- [Excel Exporting](#)

Globalization and localization in React Pivotview component

Globalization is the combination of Internationalization and localization. You can adapt the component to various languages by parsing and formatting the date or number ([Internationalization](#)) & adding culture specific customization and translation to the text ([Localization](#)).

Load CLDR-Data to the application

- Open command prompt in your machine.
- Run the following command in command prompt.

,

cd /d 'Folder path of your react application'

For example: cd /d E:\react\WebApplication

,

- Make sure that you have installed **Node and NPM** in your machine before installing the **CLDR-Data**.
- To check if you have **Node**, run this command in command prompt.

node -v

- If **Node** is not installed, then you can download and install the **Node** from this [location](#).
- To confirm if you have **NPM**, run this command in command prompt.

npm -v

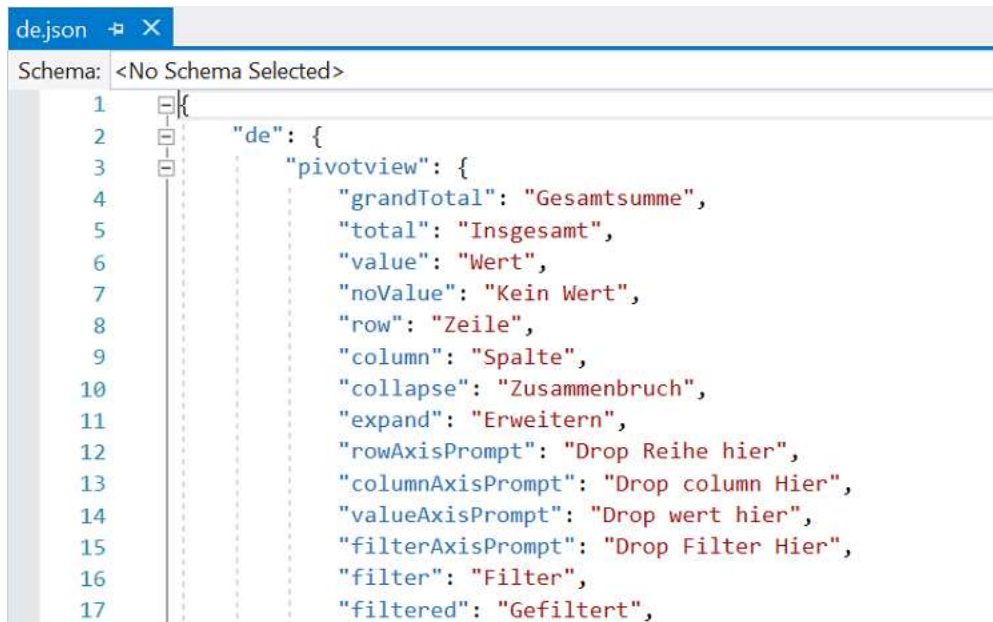
- Now, install the **CLDR-Data** package by using the following command (it installs the CLDR JSON data). To

learn more about CLDR-Data, refer to [CLDR-Data](#).

npm install cldr-data --save

- After installing the package, you can find the culture specific JSON data under the location **/node_modules/cldr-data**. Then, copy the **cldr-data** folder into your react application.
- Download the required locale packages to render the react Pivot Table component with specified locale. To download the locale definition of react components, use this [link](#).
- After downloading the ej2-locale package, copy the ej2-locale folder with required local definition file into your react application. By default, the ej2-locale package contains the localized text for static text present in components like grid, chart, pivot table, tools, and more.

The locale JSON file will look like:



Internationalization

The Internationalization library is used to globalize number, date, and time values in pivot table component using the `dataSourceSettings.formatSettings` option. In the below code sample, we set the culture and currency using the `load`, `setCulture` and `setCurrencyCode` methods. By default, pivot table component is displayed in english culture.

- Set the culture by using the `locale` property.

INDEX.JSX

```

import { FieldList, Inject, PivotViewComponent, GroupingBar, CalculatedField
} from '@syncfusion/ej2-react-pivotview';
import { L10n, setCulture, setCurrencyCode } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
setCulture('de');
setCurrencyCode('EUR');
L10n.load({
  'de-DE': {
    'pivotview': {
      'grandTotal': 'Gesamtsumme',
      'total': 'Insgesamt',
      'value': 'Wert',
      'noValue': 'Kein Wert',
      'row': 'Zeile',
      'column': 'Spalte',
      'collapse': 'Zusammenbruch',
      'expand': 'Erweitern'
    },
    'pivotfieldlist': {
      'fieldList': 'Feld Liste',
      'dropRowPrompt': 'Drop Reihe hier',
      'dropColPrompt': 'Drop column Hier',

```

```

        'dropValPrompt': 'Drop wert hier',
        'dropFilterPrompt': 'Drop Filter Hier',
        'addPrompt': 'Feld hinzufügen',
        'centerHeader': 'Ziehen Sie die Felder zwischen den Bereichen
unten:',
        'add': 'Hinzufügen',
        'drag': 'Ziehen',
        'filters': 'Filter',
        'rows': 'Zeilen',
        'columns': 'Spalten',
        'values': 'Werte',
        'error': 'Fehler',
        'dropAction': 'Berechnetes Feld nicht in jeder anderen Region
außer Wert Achse sein.',
        'search': 'Suche',
        'close': 'Schließen',
        'cancel': 'Abbrechen',
        'delete': 'Löschen',
        'alert': 'Warnung',
        'warning': 'Warnung',
        'ok': 'OK',
        'allFields': 'Alle Felder',
        'noMatches': 'Keine Treffer'
    }
}
});
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C2', useGrouping: false,
            minimumSignificantDigits: 1, maximumSignificantDigits: 3,
currency: 'EUR' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} locale='de-DE' dataSourceSettings={dataSourceSettings}
showFieldList={true} showGroupingBar={true}
allowCalculatedField={true}><Inject services={[FieldList, GroupingBar,
CalculatedField]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent,
GroupingBar, CalculatedField } from '@syncfusion/ej2-react-pivotview';
import { L10n, setCulture, setCurrencyCode } from '@syncfusion/ej2-base';

```

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
setCulture('de');
setCurrencyCode('EUR');
L10n.load({
  'de-DE': {
    'pivotview': {
      'grandTotal': 'Gesamtsumme',
      'total': 'Insgesamt',
      'value': 'Wert',
      'noValue': 'Kein Wert',
      'row': 'Zeile',
      'column': 'Spalte',
      'collapse': 'Zusammenbruch',
      'expand': 'Erweitern'
    },
    'pivotfieldlist': {
      'fieldList': 'Feld Liste',
      'dropRowPrompt': 'Drop Reihe hier',
      'dropColPrompt': 'Drop column Hier',
      'dropValPrompt': 'Drop wert hier',
      'dropFilterPrompt': 'Drop Filter Hier',
      'addPrompt': 'Feld hinzufügen',
      'centerHeader': 'Ziehen Sie die Felder zwischen den Bereichen
unten:',
      'add': 'Hinzufügen',
      'drag': 'Ziehen',
      'filters': 'Filter',
      'rows': 'Zeilen',
      'columns': 'Spalten',
      'values': 'Werte',
      'error': 'Fehler',
      'dropAction': 'Berechnetes Feld nicht in jeder anderen Region
außer Wert Achse sein.',
      'search': 'Suche',
      'close': 'Schließen',
      'cancel': 'Abbrechen',
      'delete': 'Löschen',
      'alert': 'Warnung',
      'warning': 'Warnung',
      'ok': 'OK',
      'allFields': 'Alle Felder',
      'noMatches': 'Keine Treffer'
    }
  }
});
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C2', useGrouping: false,
      minimumSignificantDigits: 1, maximumSignificantDigits: 3,
currency: 'EUR' }],

```

```

    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} locale='de-DE' dataSourceSettings={dataSourceSettings}
showFieldList={true} showGroupingBar={true}
allowCalculatedField={true}><Inject services={[FieldList, GroupingBar,
CalculatedField]} /></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

* In the above sample, **Amount** field is formatted by [NumberFormatOptions](#). For date formats, the value strings are formatted by [DateFormatOptions](#).

* By default, **locale** value is **en-US**. If you want to change the **en-US** culture to a different culture, you have to change the **locale** accordingly.

* Also, you will find more details about support format string for number formats and data formats [here](#).

<!-- markdownlint-disable MD009 -->

Decimal separators

The decimal separators of pivot table values varies based on the culture applied to the component. The culture can be set by calling the method [setCulture](#) with appropriate culture string as its parameter.

The following example demonstrates the decimal separators in **Deutsch** culture.

INDEX.JSX

```

import { FieldList, Inject, PivotViewComponent, GroupingBar, CalculatedField
} from '@syncfusion/ej2-react-pivotview';
import { loadCldr, L10n, setCulture, setCurrencyCode } from
 '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import * as currencies from './currencies.json';
import * as cagregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
import * as numberingSystems from './numberingSystems.json';
loadCldr(currencies, cagregorian, numbers, timeZoneNames, numberingSystems);
setCulture('de');
setCurrencyCode('EUR');
L10n.load({
  'de-DE': {
    'pivotview': {
      'grandTotal': 'Gesamtsumme',
      'total': 'Insgesamt',
      'value': 'Wert',
      'noValue': 'Kein Wert',

```



```

        'row': 'Zeile',
        'column': 'Spalte',
        'collapse': 'Zusammenbruch',
        'expand': 'Erweitern'
    },
    "pivotfieldlist": {
        'fieldList': 'Feld Liste',
        'dropRowPrompt': 'Drop Reihe hier',
        'dropColPrompt': 'Drop column Hier',
        'dropValPrompt': 'Drop wert hier',
        'dropFilterPrompt': 'Drop Filter Hier',
        'addPrompt': 'Feld hinzufügen',
        'centerHeader': 'Ziehen Sie die Felder zwischen den Bereichen
    unten:',
        'add': 'Hinzufügen',
        'drag': 'Ziehen',
        'filters': 'Filter',
        'rows': 'Zeilen',
        'columns': 'Spalten',
        'values': 'Werte',
        'error': 'Fehler',
        'dropAction': 'Berechnetes Feld nicht in jeder anderen Region
    außer Wert Achse sein.',
        'search': 'Suche',
        'close': 'Schließen',
        'cancel': 'Abbrechen',
        'delete': 'Löschen',
        'alert': 'Warnung',
        'warning': 'Warnung',
        'ok': 'OK',
        'allFields': 'Alle Felder',
        'noMatches': 'Keine Treffer'
    }
    }
});
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
    'Quarter' }],
        dataSource: pivotData,
        formatSettings: [{ name: 'Amount', format: 'C2', currency: 'EUR' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
    caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
    height={350} locale='de-DE' dataSourceSettings={dataSourceSettings}
    showFieldList={true} showGroupingBar={true}
    allowCalculatedField={true}><Inject services={[FieldList, GroupingBar,
    CalculatedField]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent,
GroupingBar, CalculatedField } from '@syncfusion/ej2-react-pivotview';
import { loadCldr, L10n, setCulture, setCurrencyCode, Ajax } from
 '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import * as currencies from './currencies.json';
import * as cagregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
import * as numberingSystems from './numberingSystems.json';
loadCldr(currencies, cagregorian, numbers, timeZoneNames, numberingSystems);
setCulture('de');
setCurrencyCode('EUR');
L10n.load({
  'de-DE': {
    'pivotview': {
      'grandTotal': 'Gesamtsumme',
      'total': 'Insgesamt',
      'value': 'Wert',
      'noValue': 'Kein Wert',
      'row': 'Zeile',
      'column': 'Spalte',
      'collapse': 'Zusammenbruch',
      'expand': 'Erweitern'
    },
    "pivotfieldlist": {
      'fieldList': 'Feld Liste',
      'dropRowPrompt': 'Drop Reihe hier',
      'dropColPrompt': 'Drop column Hier',
      'dropValPrompt': 'Drop wert hier',
      'dropFilterPrompt': 'Drop Filter Hier',
      'addPrompt': 'Feld hinzufügen',
      'centerHeader': 'Ziehen Sie die Felder zwischen den Bereichen
unten:',
      'add': 'Hinzufügen',
      'drag': 'Ziehen',
      'filters': 'Filter',
      'rows': 'Zeilen',
      'columns': 'Spalten',
      'values': 'Werte',
      'error': 'Fehler',
      'dropAction': 'Berechnetes Feld nicht in jeder anderen Region
außer Wert Achse sein.',
      'search': 'Suche',
      'close': 'Schließen',
      'cancel': 'Abbrechen',
      'delete': 'Löschen',
      'alert': 'Warnung',
      'warning': 'Warnung',
      'ok': 'OK',
      'allFields': 'Alle Felder',
      'noMatches': 'Keine Treffer'
    }
  }
});

```

```

    }
  });
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    formatSettings: [{ name: 'Amount', format: 'C2', currency: 'EUR' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} locale='de-DE' dataSourceSettings={dataSourceSettings}
showFieldList={true} showGroupingBar={true}
allowCalculatedField={true}><Inject services={[FieldList, GroupingBar,
CalculatedField]} /></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Localization

The [Localization](#) library allows you to localize default text content of the pivot table. The pivot table component has static text on some features (like drop area text, pivot field list title, etc...) that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the `locale` value and translation object.

The following list of properties and its values are used in the pivot table.

Locale keywords | Text

staticFieldList | Pivot Field List

fieldList | Field List

dropFilterPrompt | Drop filter here

dropColPrompt | Drop column here

dropRowPrompt | Drop row here

dropValPrompt | Drop value here

addPrompt | Add field here

adaptiveFieldHeader | Choose field

centerHeader | Drag fields between axes below:

add | add

drag | Drag

filter | Filter

filtered | Filtered

sort | Sort

remove | Remove

filters | Filters

rows | Rows

columns | Columns

values | Values

CalculatedField | Calculated Field

createCalculatedField | Create Calculated Field

fieldName | Enter the field name

error | Error

invalidFormula | Invalid formula.

dropText | Example: ('Sum(*OrderCount*)' + 'Sum(*InStock*)') * 250

dropTextMobile | Add fields and edit formula here.

dropAction | Calculated field cannot be place in any other region except value axis.

search | Search

close | Close

cancel | Cancel

delete | Delete

alert | Alert

warning | Warning

ok | OK

Sum | Sum

Avg | Avg

Count | Count

Min | Min

Max | Max

allFields | All Fields

formula | Formula

fieldExist | A field already exists in this name. Please enter a different name.

confirmText | A calculation field already exists in this name. Do you want to replace it?

noMatches | No matches

format | Summaries values by

edit | Edit

clear | Clear

formulaField | Drag and drop fields to formula

dragField | Drag field to formula

clearFilter | Clear

by | by

enterValue | Enter value

chooseDate | Enter date

all | All

multipleItems | Multiple items

Equals | Equals

DoesNotEquals | Does Not Equal

BeginWith | Begins With

DoesNotBeginWith | Does Not Begin With

EndsWith | Ends With

DoesNotEndsWith | Does Not End With

Contains | Contains

DoesNotContains | Does Not Contain

GreaterThan | Greater Than

GreaterThanOrEqualTo | Greater Than Or Equal To

LessThan | Less Than

LessThanOrEqualTo | Less Than Or Equal To

Between | Between

NotBetween | Not Between

Before | Before

BeforeOrEqualTo | Before Or Equal To

After | After

AfterOrEqualTo | After Or Equal To

member | Member

label | Label

date | Date

value | Value

labelTextContent | Show the items for which the label

dateTextContent | Show the items for which the date

valueTextContent | Show the items for which
And | and
DistinctCount | Distinct Count
Product | Product
Index | Index
SampleStDev | Sample StDev
PopulationStDev | Population StDev
SampleVar | Sample Var
PopulationVar | Population Var
RunningTotals | Running Totals
DifferenceFrom | Difference From
PercentageOfDifferenceFrom | % of Difference From
PercentageOfGrandTotal | % of Grand Total
PercentageOfColumnTotal | % of Column Total
PercentageOfRowTotal | % of Row Total
PercentageOfParentTotal | % of Parent Total
PercentageOfParentColumnTotal | % of Parent Column Total
PercentageOfParentRowTotal | % of Parent Row Total
Years | Years
Quarters | Quarters
Months | Months
Days | Days
Hours | Hours
Minutes | Minutes
Seconds | Seconds
apply | APPLY
valueFieldSettings | Value field settings
sourceName | Field name :
sourceCaption | Field caption :
summarizeValuesBy | Summarize values by :
baseField | Base field :
baseItem | Base item :
example | e.g:

editorDataLimitMsg | more items. Search to refine further.

deferLayoutUpdate | Defer Layout Update

null | null

undefined | undefined

groupOutOfRange | Out of Range

fieldDropErrorAction | The field you are moving cannot be placed in that area of the report

MoreOption | More...

memberType | Field Type

selectedHierarchy | Parent Hierarchy

formatString | Format String

expressionField | Expression

olapDropText | Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

customFormat | Enter custom format string

Measure | Measure

Dimension | Dimension

Standard | Standard

Currency | Currency

Percent | Percent

Custom | Custom

blank | (Blank)

fieldTooltip | Drag and drop fields to create an expression. And, if you want to edit the existing the calculated fields! You can achieve it by simply selecting the field under 'Calculated Members'.

fieldTitle | Field Name

QuarterYear | Quarter Year

caption | Field Caption

copy | Copy

group | Group

numberFormatString | Example: C, P, 0000 %, ###0.##0#, etc.

sortAscending | Sort ascending order

sortDescending | Sort descending order

sortNone | Sort data order

clearCalculatedField | Clear edited field info

editCalculatedField | Edit calculated field

selectGroup | Select groups

of | of

removeCalculatedField | Are you sure you want to delete this calculated field?

yes | Yes

no | No

None | None

Note: To find the latest localization keywords of pivotview and pivotfieldlist for different languages, visit this [GitHub](#) repository.

Loading Translations

To load translation object in an application, use [load](#) function of the [L10n](#) class.

The following example demonstrates the pivot table in **Deutsch** culture.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent, GroupingBar, CalculatedField } from '@syncfusion/ej2-react-pivotview';
import { L10n } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
L10n.load({
  'de-DE': {
    'pivotview': {
      'grandTotal': 'Gesamtsumme',
      'total': 'Insgesamt',
      'value': 'Wert',
      'noValue': 'Kein Wert',
      'row': 'Zeile',
      'column': 'Spalte',
      'collapse': 'Zusammenbruch',
      'expand': 'Erweitern'
    },
    'pivotfieldlist': {
      'fieldList': 'Feld Liste',
      'dropRowPrompt': 'Drop Reihe hier',
      'dropColPrompt': 'Drop column Hier',
      'dropValPrompt': 'Drop wert hier',
      'dropFilterPrompt': 'Drop Filter Hier',
      'addPrompt': 'Feld hinzufügen',
      'centerHeader': 'Ziehen Sie die Felder zwischen den Bereichen
unten:',
      'add': 'Hinzufügen',
      'drag': 'Ziehen',
      'filters': 'Filter',
      'rows': 'Zeilen',
      'columns': 'Spalten',
      'values': 'Werte',
      'error': 'Fehler',
      'dropAction': 'Berechnetes Feld nicht in jeder anderen Region
außer Wert Achse sein.',
      'search': 'Suche',
```



```

        'close': 'Schließen',
        'cancel': 'Abbrechen',
        'delete': 'Löschen',
        'alert': 'Warnung',
        'warning': 'Warnung',
        'ok': 'OK',
        'allFields': 'Alle Felder',
        'noMatches': 'Keine Treffer'
    }
}
});
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} locale='de-DE' dataSourceSettings={dataSourceSettings}
showFieldList={true} showGroupingBar={true}
allowCalculatedField={true}><Inject services={[FieldList, GroupingBar,
CalculatedField]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent,
GroupingBar, CalculatedField } from '@syncfusion/ej2-react-pivotview';
import { L10n } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
L10n.load({
    'de-DE': {
        'pivotview': {
            'grandTotal': 'Gesamtsumme',
            'total': 'Insgesamt',
            'value': 'Wert',
            'noValue': 'Kein Wert',
            'row': 'Zeile',
            'column': 'Spalte',
            'collapse': 'Zusammenbruch',
            'expand': 'Erweitern'
        },
        'pivotfieldlist': {
            'fieldList': 'Feld Liste',

```

```

        'dropRowPrompt': 'Drop Reihe hier',
        'dropColPrompt': 'Drop column Hier',
        'dropValPrompt': 'Drop wert hier',
        'dropFilterPrompt': 'Drop Filter Hier',
        'addPrompt': 'Feld hinzufügen',
        'centerHeader': 'Ziehen Sie die Felder zwischen den Bereichen
unten:',
        'add': 'Hinzufügen',
        'drag': 'Ziehen',
        'filters': 'Filter',
        'rows': 'Zeilen',
        'columns': 'Spalten',
        'values': 'Werte',
        'error': 'Fehler',
        'dropAction': 'Berechnetes Feld nicht in jeder anderen Region
außer Wert Achse sein.',
        'search': 'Suche',
        'close': 'Schließen',
        'cancel': 'Abbrechen',
        'delete': 'Löschen',
        'alert': 'Warnung',
        'warning': 'Warnung',
        'ok': 'OK',
        'allFields': 'Alle Felder',
        'noMatches': 'Keine Treffer'
    }
}
});
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} locale='de-DE' dataSourceSettings={dataSourceSettings}
showFieldList={true} showGroupingBar={true}
allowCalculatedField={true}><Inject services={[FieldList, GroupingBar,
CalculatedField]} /></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Right-to-left (RTL)

Right-to-left (RTL) provides an option to switch the text direction and layout of the pivot table component from right to left. It improves the user experiences and accessibility for users who use right-

to-left languages (Arabic, Farsi, Urdu, etc...). To enable RTL pivot table, set the `enableRtl` property to `true`.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent, GroupingBar, CalculatedField }
  from '@syncfusion/ej2-react-pivotview';
import { L10n } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
L10n.load({
  'ar-AE': {
    'pivotview': {
      'grandTotal': 'المجموع الكلي',
      'total': 'المجموع',
      'value': 'القيمة',
      'noValue': 'لا قيمة لها',
      'row': 'صف',
      'column': 'العمود',
      'collapse': 'الانهيـار',
      'expand': 'توسيع'
    },
    'pivotfieldlist': {
      'fieldList': 'قائمة الحقول',
      'dropRowPrompt': 'تراجع الخلاف هنا',
      'dropColPrompt': 'انخفاض العمود هنا',
      'dropValPrompt': 'انخفاض قيمة هنا',
      'dropFilterPrompt': 'انخفاض هنا عامل التصفية',
      'addPrompt': 'اضافة حقل هنا',
      'adaptiveFieldHeader': 'اختر الحقل',
      'centerHeader': ': اسحب المجالات بين المناطق الموضحة ادناه',
      'add': 'اضافة',
      'drag': 'اسحب',
      'filters': 'عوامل التصفية',
      'rows': 'الصفوف',
      'columns': 'الاعمدة',
      'values': 'قيم',
      'search': 'البحث',
      'close': 'قريب',
      'cancel': 'الغاء',
      'delete': 'احذف',
      'alert': 'حالة تاهب قصوى',
      'warning': 'تحذير',
      'ok': 'موافق',
      'allFields': 'جميع الحقول',
      'noMatches': 'لا مباريات'
    }
  }
});
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
```

```

    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enableRtl={true} locale='ar-AE'
dataSourceSettings={dataSourceSettings} showFieldList={true}
showGroupingBar={true} allowCalculatedField={true}><Inject
services={[FieldList, GroupingBar,
CalculatedField]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent,
GroupingBar, CalculatedField } from '@syncfusion/ej2-react-pivotview';
import { L10n } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
L10n.load({
  'ar-AE': {
    'pivotview': {
      'grandTotal': 'المجموع الكلي',
      'total': 'المجموع',
      'value': 'القيمة',
      'noValue': 'لا قيمة لها',
      'row': 'صف',
      'column': 'العمود',
      'collapse': 'الانهيـار',
      'expand': 'توسيع'
    },
    'pivotfieldlist': {
      'fieldList': 'قائمة الحقول',
      'dropRowPrompt': 'تراجع الخلاف هنا',
      'dropColPrompt': 'انخفاض العمود هنا',
      'dropValPrompt': 'انخفاض قيمة هنا',
      'dropFilterPrompt': 'انخفاض هنا عامل التصفية',
      'addPrompt': 'اضافة حقل هنا',
      'adaptiveFieldHeader': 'اختر الحقل',
      'centerHeader': 'اسحب المجالات بين المناطق الموضحة ادناه',
      'add': 'اضافة',
      'drag': 'اسحب',
      'filters': 'عوامل التصفية',
      'rows': 'الصفوف',
      'columns': 'الاعمدة',
      'values': 'قيم',
      'search': 'البحث',
      'close': 'قريب',
      'cancel': 'الغاء',
      'delete': 'احذف'
    }
  }
});

```

```

        'alert': 'حالة تاهب قصوى',
        'warning': 'تحذير',
        'ok': 'موافق',
        'allFields': 'جميع الحقول',
        'noMatches': 'لا مباريات'
    }
}
});
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataSet[],
        expandAll: false,
        filters: [],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} enableRtl={true} locale='ar-AE'
dataSourceSettings={dataSourceSettings} showFieldList={true}
showGroupingBar={true} allowCalculatedField={true}><Inject
services={[FieldList, GroupingBar, CalculatedField]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

See Also

- [Internationalization](#)
- [Localization](#)

Accessibility in React Pivotview component

The pivot table component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the pivot table component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

| Screen Reader Support | |

```

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| Accessibility Checker Validation |  |

| Axe-core Accessibility Validation |  |

<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

```

WAI-ARIA attributes

[WAI-ARIA](#) (Accessibility Initiative – Accessible Rich Internet Applications) defines a way to increase the accessibility of web pages, dynamic content, and user interface components developed with Ajax, HTML, JavaScript, and related technologies. ARIA provides additional semantics to describe the role, state, and functionality of web components. The following ARIA attributes are used in the pivot table component:

Attributes	Purpose
---	---
<code>role=grid</code>	Attribute added to identify the grid component element within the pivot table element.
<code>role=region</code>	Attribute added to identify the chart component element within the pivot table element.
<code>role=button</code>	This attribute is added to the pager navigation buttons as well as the buttons in the dialog popup such as field list, calculated field, member editor, conditional formatting of pivot table component to indicate that it is a clickable element.

| **role=table** | This attribute is added to each conditional formatting style container element to denote it as a table. |

| **role=tableItems** | This attribute is added to the container element that appears inside the number formatting popup to indicate it as a table. |

| **aria-disabled** | The buttons within the dialog popups, such as field list, calculated field and member editor, will be disabled based on their usability. To indicate its disabled state, we will add this attribute with the values **true**. By default, the attribute value is set to **false**. |

| **aria-label** | This attribute is added to label elements that are placed inside the pager, member editor popup, and calculated field popup to identify them as label elements. |

| **aria-selected** | This attribute is added to the selected treeview item in the calculated field popup with the value as **true** to denote that it is a selected element. |

| **aria-colspan** | This attribute is added to the **th** elements in the **e-table**, which represent the column span value. |

| **aria-rowspan** | This attribute is added to the **th** elements in the **e-table**, which represent the row span value. |

| **data-type** | This attribute is added to the treeview item in the calculated field popup, as well as the buttons in the grouping bar and field list. It represents the aggregate type for the specified field. |

| **data-caption** | This attribute is added to the treeview item in the calculated field popup, as well as the buttons in the grouping bar and field list. It represents the caption for the specified field. |

| **data-basefield** | This attribute is added to the treeview item in the calculated field popup, as well as the buttons in the grouping bar and field list. It denotes the base field for the specified field, which is used to display the values for aggregation types such as **DifferenceFrom**, **PercentageOfDifferenceFrom**, and **PercentageOfParentTotal**. |

| **data-baseitem** | This attribute is added to the treeview item in the calculated field popup, as well as the buttons in the grouping bar and field list. It denotes the base item for the specified field, which is used to display the values for aggregation types such as **DifferenceFrom**, **PercentageOfDifferenceFrom**, and **PercentageOfParentTotal**. |

| **data-field** | This attribute is added to the treeview item in the calculated field popup. It denotes the name of the specified field. |

| **data-membertype** | This attribute is added to the treeview item in the calculated field popup. It denotes the member type of the selected OLAP calculated field. |

| **data-hierarchy** | This attribute is added to the treeview item in the calculated field popup. It denotes the parent hierarchy unique name of the selected OLAP calculated field. |

| **data-formula** | This attribute is added to the treeview item in the calculated field popup. It denotes the formula used for the specified calculated field. |

| **data-formatString** | This attribute is added to the treeview item in the calculated field popup. It denotes the format string used for the specified calculated field. |

| **data-customformatstring** | This attribute is added to the treeview item in the calculated field popup. It denotes the custom format string used for the specified calculated field. |

Keyboard interaction

The pivot table component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Message component.

| **Press** | **To do this** |

| --- | --- |

| Tab / Shift + Tab | To focus the close icon in the message. |

| Enter / Space | Closes the focused close icon's message. |

Pivot Table

| **Press** | **To do this** |

| --- | --- |

| Tab | Moves the cell focus right side. If no cells are focused, it moves to the next active element in the browser page. |

| Shift + Tab | Moves the cell focus left side. If no cells are focused, it moves to the previous active element in the browser page. |

| DownArrow | Moves the cell focus downwards. If the selection is enabled in the pivot table, then it will move downwards to select next row or column or individual cell. |

| UpArrow | Moves the cell focus upwards. If the selection is enabled in the pivot table, then it will move upwards to select previous row or column or individual cell. |

| LeftArrow | Moves the cell focus left side. If the selection is enabled in the pivot table, then it will move left side to select previous row or column or individual cell. |

| RightArrow | Moves the cell focus right side. If the selection is enabled in the pivot table, then it will move right side to select next row or column or individual cell. |

| Shift + DownArrow | Extends the cell selection downwards. |

| Shift + UpArrow | Extends the cell selection selection upwards. |

| Shift + LeftArrow | Extends the cell selection to the left side. |

| Shift + RightArrow | Extends the cell selection to the right side. |

| Ctrl + A | Selects all cells. |

| Esc | Deselects all cells. If the current active element is a context menu, then the context menu popup will be closed. |

| Home | Goes to the first cell in the current row. |

| End | Goes to the last cell in the current row. |

| Ctrl + Home | Goes to the first cell in the table. |

| Ctrl + End | Goes to the last cell in the table. |

| Enter | If the current cell is an expand/collapse cell, it performs expand/collapse operation (drill operation). If the current row/column header is in value sort state, it performs value sorting. If the current cell is in selection state, it moves to the next row, column or individual cell. If drill-through or editing is enabled in the pivot table, the drill-through dialog will be opened based on the selected value cell. If the current active element is a context menu popup, menu selection will be performed. |

| Shift + Enter | If value sorting is enabled in the pivot table and the current cell is a header with respect to its value axis, it performs value sorting to either ascending or descending order. If the current cell is in selection state, it moves to the previous row, column or individual cell. |

| Ctrl + Enter | If hyperlink is enabled in the current cell, it performs hyperlink selection. |

| Shift + F10 or Menu | If context menu is enabled in the pivot table, the context menu popup will be opened in the current cell. |

Field List

| Press | To do this |

| --- | --- |

| Shift + Ctrl + F | If the popup field list is enabled in either the pivot table or the pivot chart, the field list dialog will be opened. |

| Tab | Moves to the next active element in the field list. If no active elements present, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element in the field list. If no active elements present, it moves to the previous active element in the browser page. |

| Shift + F | If the current active element is a field's button and if it has a filter icon, the filter dialog will open to perform filtering. |

| Shift + S | If the current active element is a field's button and if it has a sort icon, the sorting will be performed to the selected field. |

| Shift + E | If the current active element is a calculated field's button and if it has an edit icon, the calculated field dialog will be opened to perform editing the selected calculated field. |

| Enter | Performs the selection operation of the current active element. If the current active element is a field's button and it has a dropdown icon, the aggregation menu will open to perform calculations using aggregation options to the selected value field. |

| Delete | If the current active element is a field's button, the selected field will be removed from the current report. |

| DownArrow | If the current active element is a tree node, it moves to the next node. |

| UpArrow | If the current active element is a tree node, it moves to the previous node. |

| LeftArrow | If the current active element is a tree node, it collapses the current node. |

| RightArrow | If the current active element is a tree node, it expands the current node. |

| Home | If the current active element is a tree node, it goes to the first node. |

| End | If the current active element is a tree node, it goes to the last node. |

| Space | If the current active element is a tree node or a checkbox element, it will be either checked or unchecked. |

| Esc or Escape | Closes the popup field list dialog. |

Grouping Bar

| Press | To do this |

| --- | --- |

| Tab | Moves to the next active element (field's button) in the grouping bar. If no active elements present, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element (field's button) in the grouping bar. If no active elements present, it moves to the previous active element in the browser page. |

| Shift + F | If the current active element is a field's button and if it has a filter icon, the filter dialog will be opened to perform filtering. |

| Shift + S | If the current active element is a field's button and if it has a sort icon, the sorting will be performed to the selected field. |

| Shift + E | If the current active element is a calculated field's button and if it has an edit icon, the calculated field dialog will be opened to perform editing the selected calculated field. |

| Enter | Performs the selection operation of the current active element. If the current active element is a field's button and if it has a dropdown icon, the aggregation menu will be opened to perform calculations using aggregation options to the selected value field. |

| Delete | If the current active element is a field's button, the selected field will be removed from the current report. |

| DownArrow | If the current active element is a dropdown list, the next item will be selected. |

| UpArrow | If the current active element is a dropdown list, the previous item will be selected. |

| Home | If the current active element is a dropdown list, the first item will be selected. |

| End | If the current active element is a dropdown list, the last item will be selected. |

| Alt + Down | If the current active element is a dropdown list, the popup will be opened. |

| Alt + Down | If the current active element is a dropdown list, the popup will be closed. |

| Esc or Escape | Closes the dropdown list.

Filter Dialog

| Press | To do this |

| --- | --- |

| Shift + F | If the current active element is a field's button and if it has a filter icon in either the field list or grouping bar UI, the filter dialog will be opened to perform filtering. |

| Tab | Moves to the next active element in the filter dialog. If no active elements present, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element in the filter dialog. If no active elements present, it moves to the previous active element in the browser page. |

| DownArrow | If the current active element is a tree node, it moves to the next node. |

| UpArrow | If the current active element is a tree node, it moves to the previous node. |

| LeftArrow | If the current active element is a tree node, it collapses the current node. If the current active element is a tab, it moves focus to the previous tab element. |

| RightArrow | If the current active element is a tree node, it expands the current node. If the current active element is a tab, it moves focus to the next tab element. |

| Home | If the current active element is a tree node, it goes to the first node. |

| End | If the current active element is a tree node, it goes to the last node. |

| Space | If the current active element is a tree node or a checkbox element, it will be either checked or unchecked. |

| Alt + Down | If the current active element is a DropDownList or DatePicker or DateTimePicker, the popup will be opened. |

| Alt + Up | If the current active element is a DropDownList or DatePicker or DateTimePicker, the popup will be closed. |

| Enter | Performs the selection operation of the current active element. If the current active element is a tab, the current tab element will be selected. If the current active element is a tree node, the current node will be either checked or unchecked. If the current active element is DropDownList, the focus item will be selected, and the popup list will close when it is open. Otherwise, toggles the popup list. |

| Esc or Escape | Closes the filter dialog. |

Calculated Field Dialog

| Press | To do this |

| --- | --- |

| Shift + E | If the current active element is a field's button and if it has an edit icon in either the field list or grouping bar UI, the calculated field dialog will be opened to perform editing the selected calculated field. |

| Tab | Moves to the next active element in the calculated field dialog. If no active elements present, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element in the calculated field dialog. If no active elements present, it moves to the previous active element in the browser page. |

| DownArrow | If the current active element is a tree node, it moves to the next node. |

| UpArrow | If the current active element is a tree node, it moves to the previous node. |

| LeftArrow | If the current active element is a tree node, it collapses the current node. |

| RightArrow | If the current active element is a tree node, it expands the current node. If the current active element is a tree node and has a menu icon, the aggregation menu will be opened to select appropriate aggregation type to the selected field. |

| Home | If the current active element is a tree node, it goes to the first node. |

| End | If the current active element is a tree node, it goes to the last node. |

| Enter | Performs the selection operation of the current active element. If the current active element is a tree node, it copies the selected field name/formula to the formula text area to perform calculations. |

| Esc or Escape | Closes the calculated field dialog. |

Formatting Dialog

| Press | To do this |

| --- | --- |

| Tab | Moves to the next active element in the formatting dialog. If no active elements present, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element in the formatting dialog. If no active elements present, it moves to the previous active element in the browser page. |

| DownArrow | If the current active element is a DropDownList, the next item will be selected. |

| UpArrow | If the current active element is a DropDownList, the previous item will be selected. |

| Home | If the current active element is a DropDownList, the first item will be selected. |

| End | If the current active element is a DropDownList, the last item will be selected. |

| Alt + Down | If the current active element is a DropDownList or ColorPicker, the popup will be opened. |

| Alt + Down | If the current active element is a DropDownList or ColorPicker, the popup will be closed. |

| Enter | Performs the selection operation of the current active element. |

| Esc or Escape | Closes the formatting dialog. |

Toolbar

| Press | To do this |

| --- | --- |

| Tab | Moves to the next active option in the toolbar. If no active elements present, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active option in the toolbar. If no active elements present, it moves to the previous active element in the browser page. |

| Enter | Performs the selection operation of the current active element. |

Drill-Through Dialog

| Press | To do this |

| --- | --- |

| Tab | Moves to the next active element in the drill-through dialog. If the current active element is a Grid cell, it moves the cell focus to right side. If no active elements present, then it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element in the drill-through dialog. If the current active element is a Grid cell, it moves the cell focus to left side, If no active elements present, then it moves to the previous active element in the browser page. |

| DownArrow | Moves the row/cell focus downwards. |

| UpArrow | Moves the row/cell focus upwards. |

| LeftArrow | Moves the cell focus left side. |

| RightArrow | Moves the cell focus right side. |

| Home | Goes to the first cell in the current row. |

| End | Goes to the last cell in the current row. |

| Ctrl + Home | Goes to the first cell in the table. |

| Ctrl + End | Goes to the last cell in the table. |

| Enter | Performs the selection operation of the current active element. |

| Esc or Escape | If the cell is in selected state, then it deselects all rows/cells. If the row/cell is in edit state, it cancels the current entries in the row/cell. If the current active element is not a row/cell, it closes the drill-through dialog. |

| F2 | Initiate editing a row/cell in the data grid. |

| Insert | Adds a new row/cell in the data grid. |

| Delete | Removes the selected row in the data grid. |

Some commonly used applicable key combinations and their relative functionalities in all dialogs are listed below.

| Press | To do this |

| --- | --- |

| Tab | Moves to the next active element in the dialog. If either no active elements present in the dialog or an overlay is not present in the dialog, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element in the dialog. If either no active elements present in the dialog or an overlay is not present in the dialog, it moves to the previous active element in the browser page. |

| Space | If the current active element is a tree node or a checkbox element, it will be either checked or unchecked. |

| Enter | When the Dialog button or any input (except text area) is in focus state, when pressing the Enter key, the click event associated with the primary button or button will be triggered. The Enter key will not be worked, when the dialog content contains any text area with initial focus. |

| Esc or Escape | Closes the dialog. |

Ensuring accessibility

The pivot table component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the pivot table component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the pivot table component with accessibility tools.

INDEX.JSX

```
import {
  FieldList, Inject, PivotViewComponent, CalculatedField, Toolbar,
  GroupingBar,
  ConditionalFormatting, NumberFormatting, PDFExport, ExcelExport, Grouping,
  FieldList, Inject, PivotViewComponent, CalculatedField, Toolbar,
  ConditionalFormatting, NumberFormatting, PDFExport, ExcelExport
} from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import * as data from './datasource.json';
function App() {
  let pivotData = data.pivotData;
  var dataSourceSettings = {
    dataSource: pivotData,
    expandAll: true,
    enableSorting: true,
    allowLabelFilter: true,
    allowValueFilter: true,
    sortSettings: [{ name: 'company', order: 'Descending' }],
    formatSettings: [{ name: 'balance', format: 'C' }, { name: 'date',
format: 'dd/MM/yyyy-hh:mm', type: 'date' }],
    drilledMembers: [{ name: 'product', items: ['Bike', 'Car'] }, { name:
'gender', items: ['male'] }],
    filterSettings: [
      { name: 'date', type: 'Date', condition: 'Between', value1: new
Date('02/16/2000'), value2: new Date('02/16/2002') },
      { name: 'age', type: 'Number', condition: 'Between', value1: '25',
value2: '35' },
      { name: 'eyeColor', type: 'Exclude', items: ['blue'] }
    ],
    rows: [{ name: 'state' }, { name: 'eyeColor' }],
    columns: [{ name: 'gender', caption: 'Population' }, { name: 'isActive'
}],
    values: [{ name: 'balance' }, { name: 'quantity' }],
    filters: [],
    conditionalFormatSettings: [
```

```

    {
      measure: 'balance',
      value1: 100000,
      conditions: 'LessThan',
      style: {
        backgroundColor: '#80cbc4',
        color: 'black',
        fontFamily: 'Tahoma',
        fontSize: '12px'
      }
    },
    {
      value1: 10,
      value2: 20,
      measure: 'quantity',
      conditions: 'Between',
      style: {
        backgroundColor: '#f48fb1',
        color: 'black',
        fontFamily: 'Tahoma',
        fontSize: '12px'
      }
    }
  ]
}
var pivotObj;
var toolbarOptions = [
  'New', 'Save', 'SaveAs', 'Rename', 'Remove', 'Load', 'Grid', 'Chart',
  'Export', 'SubTotal', 'GrandTotal',
  'ConditionalFormatting', 'NumberFormatting', 'FieldList'
];
var groupingSettings = {
  showFieldsPanel: true
};
var chartSettings = {
  value: 'Amount', enableExport: true, chartSeries: {
    type: 'Column', animation: { enable: false }
  }, enableMultipleAxis: false
};
var editSettings = {
  allowAdding: true, allowDeleting: true, allowEditing: true, mode:
  'Normal'
};
var gridSettings = {
  columnWidth: 140,
  contextMenuItems: [
    'Aggregate', 'CalculatedField', 'Drillthrough', 'Excel Export', 'Pdf
Export', 'Csv Export',
    'Expand', 'Collapse', 'Sort Ascending', 'Sort Descending'
  ]
}
function saveReport(args) {
  var reports = [];
  var isSaved = false;
  if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
  "") {
    reports = JSON.parse(localStorage.pivotviewReports);
  }

```

```

    }
    if (args.report && args.reportName && args.reportName !== '' &&
args.reportName !== 'Sample Report') {
        reports.map(function (item) {
            if (args.reportName === item.reportName) {
                item.report = args.report; isSaved = true;
            }
        });
        if (!isSaved) {
            reports.push(args);
        }
        localStorage.pivotviewReports = JSON.stringify(reports);
    }
}
function fetchReport(args) {
    var reportCollection = [];
    var reeportList = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item) {
    reeportList.push(item.reportName); });
    args.reportName = reeportList;
}
function loadReport(args) {
    var reportCollection = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item) {
        if (args.reportName === item.reportName) {
            args.report = item.report;
        }
    });
    if (args.report) {
        pivotObj.dataSourceSettings =
JSON.parse(args.report).dataSourceSettings;
    }
}
function removeReport(args) {
    var reportCollection= [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    for (var i = 0; i < reportCollection.length; i++) {
        if (reportCollection[i].reportName === args.reportName) {
            reportCollection.splice(i, 1);
        }
    }
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        localStorage.pivotviewReports = JSON.stringify(reportCollection);
    }
}

```



```

function renameReport(args) {
    var reportCollection= [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    if (args.isReportExists) {
        for (var i = 0; i < reportCollection.length; i++) {
            if (reportCollection[i].reportName === args.rename) {
                reportCollection.splice(i, 1);
            }
        }
    }
    reportCollection.map(function (item) {
        if (args.reportName === item.reportName) {
            item.reportName = args.rename;
        }
    });
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        localStorage.pivotviewReports = JSON.stringify(reportCollection);
    }
}

function newReport() {
    pivotObj.setProperties({ dataSource: { columns: [], rows: [], values:
    [], filters: [] } }, false);
}

function beforeToolbarRender(args) {
    args.customToolbar.splice(6, 0, {
        type: 'Separator'
    });
    args.customToolbar.splice(9, 0, {
        type: 'Separator'
    });
}

return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'}
height={450} showFieldList={true} gridSettings={gridSettings}
allowExcelExport={true}
allowConditionalFormatting={true} allowNumberFormatting={true}
allowPdfExport={true} showToolbar={true}
allowCalculatedField={true} displayOption={{ view: 'Both' }}
toolbar={toolbarOptions} newReport={newReport.bind(this)}
renameReport={renameReport.bind(this)}
removeReport={removeReport.bind(this)} loadReport={loadReport.bind(this)}
fetchReport={fetchReport.bind(this)}
saveReport={saveReport.bind(this)}
toolbarRender={beforeToolbarRender.bind(this)}
showGroupingBar={true} groupingBarSettings={groupingSettings}
chartSettings={chartSettings} editSettings={editSettings}
allowGrouping={true} allowDeferLayoutUpdate={true}>
    <Inject services=[
        FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport,
        ConditionalFormatting, NumberFormatting, GroupingBar,
        Grouping
    ] />
</PivotViewComponent>

```

```
);  
function pivotData1() {  
  var pivotData = [  
    {  
      "_id": "5a940692c2d185d9fde50e5e",  
      "index": 0,  
      "guid": "810a1191-81bd-4c18-ac73-d16ad3fc80eb",  
      "isActive": false,  
      "balance": 2430.87,  
      "advance": 7658,  
      "quantity": 11,  
      "age": 21,  
      "eyeColor": "blue",  
      "name": "Skinner Ward",  
      "gender": "male",  
      "company": "GROK",  
      "email": "skinnerward@grok.com",  
      "phone": "+1 (931) 600-3042",  
      "date": "Wed Feb 16 2000 15:01:01 GMT+0530 (India Standard Time)",  
      "product": "Flight",  
      "state": "New Jercey",  
      "pno": "FEDD2340"  
    },  
    {  
      "_id": "5a940692c5752f1ed81bbb3d",  
      "index": 1,  
      "guid": "41c9986b-ccef-459e-a22d-5458bbdca9c7",  
      "isActive": true,  
      "balance": 3192.7,  
      "advance": 6124,  
      "quantity": 15,  
      "age": 27,  
      "eyeColor": "brown",  
      "name": "Gwen Dixon",  
      "gender": "female",  
      "company": "ICOLOGY",  
      "email": "gwendixon@icology.com",  
      "phone": "+1 (951) 589-2187",  
      "date": "Sun Feb 10 1991 20:28:59 GMT+0530 (India Standard Time)",  
      "product": "Jet",  
      "state": "Vetaikan",  
      "pno": "ERTS4512"  
    },  
    {  
      "_id": "5a9406924c0e7f4c98a82ca7",  
      "index": 2,  
      "guid": "50d2bf16-9092-4202-84f6-e892721fe5a5",  
      "isActive": true,  
      "balance": 1663.84,  
      "advance": 7631,  
      "quantity": 14,  
      "age": 28,  
      "eyeColor": "green",  
      "name": "Deena Gillespie",  
      "gender": "female",  
      "company": "OVERPLEX",  
      "email": "deenagillespie@overplex.com",  
    }  
  ]  
}
```

```

    "phone": "+1 (826) 588-3430",
    "date": "Thu Mar 18 1993 17:07:48 GMT+0530 (India Standard Time)",
    "product": "Car",
    "state": "New Jercy",
    "pno": "ERTS4512"
  },
  {
    "_id": "5a940692dd9db638eee09828",
    "index": 3,
    "guid": "b8bdc65e-4338-440f-a731-810186ce0b3a",
    "isActive": true,
    "balance": 1601.82,
    "advance": 6519,
    "quantity": 18,
    "age": 33,
    "eyeColor": "green",
    "name": "Susanne Peterson",
    "gender": "female",
    "company": "KROG",
    "email": "susanpeterson@krog.com",
    "phone": "+1 (868) 499-3292",
    "date": "Sat Feb 09 2002 04:28:45 GMT+0530 (India Standard Time)",
    "product": "Jet",
    "state": "Vetaikan",
    "pno": "CCOP1239"
  },
  {
    "_id": "5a9406926f9971a87eae51af",
    "index": 4,
    "guid": "3f4c79ec-a227-4210-940f-162ca0c293de",
    "isActive": false,
    "balance": 1855.77,
    "advance": 7333,
    "quantity": 20,
    "age": 33,
    "eyeColor": "green",
    "name": "Stokes Hicks",
    "gender": "male",
    "company": "SIGNITY",
    "email": "stokeshicks@signity.com",
    "phone": "+1 (927) 585-2980",
    "date": "Fri Mar 12 2004 11:08:06 GMT+0530 (India Standard Time)",
    "product": "Van",
    "state": "Tamilnadu",
    "pno": "MEWD9812"
  },
  {
    "_id": "5a940692bcbbcdde08fcf7ec",
    "index": 5,
    "guid": "1d0ee387-14d4-403e-9a0c-3a8514a64281",
    "isActive": true,
    "balance": 1372.23,
    "advance": 5668,
    "quantity": 16,
    "age": 39,
    "eyeColor": "green",
    "name": "Sandoval Nicholson",

```

```

    "gender": "male",
    "company": "IDEALIS",
    "email": "sandovalnicholson@idealis.com",
    "phone": "+1 (951) 438-3539",
    "date": "Sat Aug 30 1975 22:02:15 GMT+0530 (India Standard Time)",
    "product": "Bike",
    "state": "Tamilnadu",
    "pno": "CCOP1239"
  },
  {
    "_id": "5a940692ff31a6e1cdd10487",
    "index": 6,
    "guid": "58417d45-f279-4e21-ba61-16943d0f11c1",
    "isActive": false,
    "balance": 2008.28,
    "advance": 7107,
    "quantity": 14,
    "age": 20,
    "eyeColor": "brown",
    "name": "Blake Thornton",
    "gender": "male",
    "company": "IMMUNICS",
    "email": "blakethornton@immunics.com",
    "phone": "+1 (852) 462-3571",
    "date": "Mon Oct 03 2005 05:16:53 GMT+0530 (India Standard Time)",
    "product": "Jet",
    "state": "New Jercey",
    "pno": "CCOP1239"
  },
  {
    "_id": "5a9406928f2f2598c7ac7809",
    "index": 7,
    "guid": "d16299e3-e243-4e57-90fb-52446c4c0275",
    "isActive": false,
    "balance": 2052.58,
    "advance": 7431,
    "quantity": 20,
    "age": 22,
    "eyeColor": "blue",
    "name": "Dillard Sharpe",
    "gender": "male",
    "company": "INEAR",
    "email": "dillardsharpe@inear.com",
    "phone": "+1 (963) 473-2308",
    "date": "Thu May 25 1978 04:57:00 GMT+0530 (India Standard Time)",
    "product": "Jet",
    "state": "Rajkot",
    "pno": "ERTS4512"
  },
  {
    "_id": "5a940692195f82585af58362",
    "index": 8,
    "guid": "a8662b61-9d66-4b99-8a47-c0375ffff4ce",
    "isActive": true,
    "balance": 2784.64,
    "advance": 7948,
    "quantity": 18,

```

```

        "age": 22,
        "eyeColor": "blue",
        "name": "Davidson Brewer",
        "gender": "male",
        "company": "PROXSOFTE",
        "email": "davidsonbrewer@proxsoft.com",
        "phone": "+1 (958) 592-3948",
        "date": "Wed Jul 14 1982 19:49:29 GMT+0530 (India Standard Time)",
        "product": "Van",
        "state": "Vetaikan",
        "pno": "FEDD2340"
      }
    ]
    return pivotData;
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import {
  FieldList, IDataOptions, Inject, PivotViewComponent, CalculatedField,
  Toolbar, RemoveReportArgs, ToolbarArgs,
  ConditionalFormatting, IDataset, RenameReportArgs, SaveReportArgs,
  FetchReportArgs,
  LoadReportArgs, NumberFormatting, PDFExport, ExcelExport, GroupingBar,
  GroupingBarSettings,
  CellEditSettings, Grouping
} from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartSettings';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridSettings';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: true,
    enableSorting: true,
    allowLabelFilter: true,
    allowValueFilter: true,
    sortSettings: [{ name: 'company', order: 'Descending' }],
    formatSettings: [{ name: 'balance', format: 'C' }, { name: 'date',
format: 'dd/MM/yyyy-hh:mm', type: 'date' }],
    drilledMembers: [{ name: 'product', items: ['Bike', 'Car'] }, { name:
'gender', items: ['male'] }],
    filterSettings: [
      { name: 'date', type: 'Date', condition: 'Between', value1: new
Date('02/16/2000'), value2: new Date('02/16/2002') },
      { name: 'age', type: 'Number', condition: 'Between', value1: '25',
value2: '35' },
      { name: 'eyeColor', type: 'Exclude', items: ['blue'] }
    ],
  },

```

```

    rows: [{ name: 'state' }, { name: 'eyeColor' }],
    columns: [{ name: 'gender', caption: 'Population' }, { name: 'isActive'
}],
    values: [{ name: 'balance' }, { name: 'quantity' }],
    filters: [],
    conditionalFormatSettings: [
      {
        measure: 'balance',
        value1: 100000,
        conditions: 'LessThan',
        style: {
          backgroundColor: '#80cbc4',
          color: 'black',
          fontFamily: 'Tahoma',
          fontSize: '12px'
        }
      },
      {
        value1: 10,
        value2: 20,
        measure: 'quantity',
        conditions: 'Between',
        style: {
          backgroundColor: '#f48fb1',
          color: 'black',
          fontFamily: 'Tahoma',
          fontSize: '12px'
        }
      }
    ]
  }
  let pivotObj: PivotViewComponent;
  let toolbarOptions: any = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
  'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'];
  let groupingSettings: GroupingBarSettings = {
    showFieldsPanel: true
  } as GroupingBarSettings;
  let chartSettings: ChartSettings = {
    value: 'Amount', enableExport: true, chartSeries: {
      type: 'Column', animation: { enable: false }
    }, enableMultipleAxis: false,
  } as ChartSettings;
  let editSettings: CellEditSettings = {
    allowEditing: true, allowAdding: true, allowDeleting: true, mode:
'Normal'
  } as CellEditSettings
  let gridSettings: GridSettings = {
    columnWidth: 140,
    contextMenuItems: [
      'Aggregate', 'CalculatedField', 'Drillthrough', 'Excel Export', 'Pdf
Export', 'Csv Export', 'Expand', 'Collapse', 'Sort Ascending', 'Sort
Descending'
    ]
  } as GridSettings;
  function saveReport(args: any): void {

```

```

let reports: SaveReportArgs[] = [];
let isSaved: boolean = false;
if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
"") {
    reports = JSON.parse(localStorage.pivotviewReports);
}
if (args.report && args.reportName && args.reportName !== '' &&
args.reportName !== 'Sample Report') {
    reports.map(function (item: any): any {
        if (args.reportName === item.reportName) {
            item.report = args.report; isSaved = true;
        }
    });
    if (!isSaved) {
        reports.push(args);
    }
    localStorage.pivotviewReports = JSON.stringify(reports);
}
}
function fetchReport(args: FetchReportArgs): void {
    let reportCollection: string[] = [];
    let reeportList: string[] = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
"") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item: any): void {
        reeportList.push(item.reportName); });
    args.reportName = reeportList;
}
function loadReport(args: LoadReportArgs): void {
    let reportCollection: string[] = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
"") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item: any): void {
        if (args.reportName === item.reportName) {
            args.report = item.report;
        }
    });
    if (args.report) {
        pivotObj.dataSourceSettings =
JSON.parse(args.report).dataSourceSettings;
    }
}
function removeReport(args: RemoveReportArgs): void {
    let reportCollection: any[] = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
"") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    for (let i: number = 0; i < reportCollection.length; i++) {
        if (reportCollection[i].reportName === args.reportName) {
            reportCollection.splice(i, 1);
        }
    }
}

```

```

    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        localStorage.pivotviewReports = JSON.stringify(reportCollection);
    }
}
function renameReport(args: RenameReportArgs): void {
    let reportCollection: any[] = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    if (args.isReportExists) {
        for (let i: number = 0; i < reportCollection.length; i++) {
            if (reportCollection[i].reportName === args.rename) {
                reportCollection.splice(i, 1);
            }
        }
    }
    reportCollection.map(function (item: any): any {
        if (args.reportName === item.reportName) {
            item.reportName = args.rename;
        }
    });
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        localStorage.pivotviewReports = JSON.stringify(reportCollection);
    }
}
function newReport(): void {
    pivotObj.setProperties({ dataSource: { columns: [], rows: [], values:
    [], filters: [] } }, false);
}
function beforeToolbarRender(args: any): void {
    args.customToolbar.splice(6, 0, {
        type: 'Separator'
    });
    args.customToolbar.splice(9, 0, {
        type: 'Separator'
    });
}
return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'}
height={350} showFieldList={true} gridSettings={gridSettings}
allowExcelExport={true}
allowConditionalFormatting={true} allowNumberFormatting={true}
allowPdfExport={true} showToolbar={true}
allowCalculatedField={true} displayOption={{ view: 'Both' }}
toolbar={toolbarOptions} newReport={newReport.bind(this)}
renameReport={renameReport.bind(this)}
removeReport={removeReport.bind(this)} loadReport={loadReport.bind(this)}
fetchReport={fetchReport.bind(this)} saveReport={saveReport.bind(this)}
toolbarRender={beforeToolbarRender.bind(this)}
showGroupingBar={true} groupingBarSettings={groupingSettings}
chartSettings={chartSettings} allowGrouping={true}
allowDeferLayoutUpdate={true} editSettings={editSettings}><Inject
services={

```



```

        FieldList, CalculatedField, Toolbar, PDFExport, ExcelExport,
        ConditionalFormatting, NumberFormatting, GroupingBar, Grouping
    ]} /></PivotViewComponent>
    );
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

See also

- [Accessibility in Syncfusion React components](#)

Ej1 api migration in React Pivotview component

This article describes the API migration process of pivot table component from Essential JS 1 to Essential JS 2.

Data Binding

{% raw %}

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Data source | **property:** dataSource

<EJ.PivotGrid id="PivotGrid" dataSource={pivotdataSource}></EJ.PivotGrid>

var pivotdataSource = {
data: []
}; | **property:** dataSourceSettings

<PivotViewComponent id='PivotView' dataSourceSettings={dataSourceSettings}></PivotViewComponent>

let dataSourceSettings: IDataOptions = {
dataSource: [] as IDataset[]
}; |

| Rows | **property:** rows

<EJ.PivotGrid id="PivotGrid" dataSource={pivotdataSource}></EJ.PivotGrid>

var pivotdataSource = {
rows: [{
fieldName: "Country", fieldCaption: "Country"}]
}; | **property:** rows

<PivotViewComponent id='PivotView' dataSourceSettings={dataSourceSettings}></PivotViewComponent>

let dataSourceSettings: IDataOptions = {
row: [{
 name: 'company', caption: 'Industry' }]
}; |

| Columns | **property:** columns

<EJ.PivotGrid id="PivotGrid" dataSource={pivotdataSource}></EJ.PivotGrid>

var pivotdataSource = {
columns: [{
fieldName: "Country", fieldCaption: "Country"}]
}; | **property:** columns

<PivotViewComponent id='PivotView' dataSourceSettings={dataSourceSettings}></PivotViewComponent>

let dataSourceSettings: IDataOptions = {
columns: [{
 name: 'company', caption: 'Industry' }]
}; |

| Values | **property:** values

<EJ.PivotGrid id="PivotGrid" dataSource={pivotdataSource}></EJ.PivotGrid>

var pivotdataSource = {
values: [{
fieldName: "balance", fieldCaption: "Balance(\$)" }]
 }; | **property:** values

<PivotViewComponent id='PivotView' dataSourceSettings={dataSourceSettings}></PivotViewComponent>

let

```

dataSourceSettings: IDataOptions = {<br/>values: [{<br/> name: 'balance', caption: 'Balance($)'
}]<br/>};|
| Filters | property: filters<br/><br/><EJ.PivotGrid id="PivotGrid" dataSource=
{pivotdataSource}></EJ.PivotGrid><br/><br/>var pivotdataSource = {<br/>filters:
[{{<br/>fieldName: "Country", fieldCaption: "Country" }}<br/>];| property:
filters<br/><br/><PivotViewComponent id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent><br/><br/>let
dataSourceSettings: IDataOptions = {<br/>filters: [{<br/> name: 'company', caption:
'Industry'}]<br/>};|
| Value axis position | Not Applicable | property: valueAxis<br/><br/><PivotViewComponent
id='PivotView' dataSourceSettings={dataSourceSettings}></PivotViewComponent><br/><br/>let
dataSourceSettings: IDataOptions = {<br/>valueAxis: 'row'<br/>};|
{% endraw %}

```

Aggregation

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

```

| Summary Type | property: summaryType<br/><br/><EJ.PivotGrid id="PivotGrid" dataSource=
{pivotdataSource}></EJ.PivotGrid><br/><br/>var pivotdataSource = {<br/>values: [{<br/>
fieldName: "balance",<br/>fieldCaption: "Balance($)",<br/>summaryType:
ej.PivotAnalysis.SummaryType.Count }}<br/>];| property: type<br/><br/><PivotViewComponent
id='PivotView' dataSourceSettings={dataSourceSettings}></PivotViewComponent><br/><br/>let
dataSourceSettings: IDataOptions = {<br/>values: [{<br/>name: 'balance', caption: 'Balance($)',
type: 'Count' }}<br/>};|

```

Number Format

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

```

| Format settings | property: format<br/><br/><EJ.PivotGrid id="PivotGrid" dataSource=
{pivotdataSource}></EJ.PivotGrid><br/><br/>var pivotdataSource = {<br/>values: [{<br/>
fieldName: "balance",<br/>fieldCaption: "Balance($)",<br/>format: "currency"
}]<br/>};| property: formatSettings<br/><br/><PivotViewComponent id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent><br/><br/>let
dataSourceSettings: IDataOptions = {<br/>formatSettings: [{<br/>name: 'balance', format: 'C'
},<br/>{ name: 'date', format: 'dd/MM/yyyy-hh:mm', type: 'date' }}<br/>};|

```

Summary Customization

{% raw %}

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

```

| Show/hide grand totals | property: enableGrandTotal<br/><br/><EJ.PivotGrid id="PivotGrid"
dataSource= {pivotdataSource}></EJ.PivotGrid><br/><br/>var pivotdataSource =

```

```
{<br/>enableGrandTotal: false<br/>}; | property: showGrandTotals<br/><br/><PivotViewComponent
id='PivotView' dataSourceSettings={dataSourceSettings}></PivotViewComponent><br/><br/>let
dataSourceSettings: IDataOptions = {<br/>showGrandTotals: false<br/>}; |
```

```
| Show/hide row grand totals | property: enableRowGrandTotal<br/><br/><EJ.PivotGrid id="PivotGrid"
dataSource= {pivotdataSource}></EJ.PivotGrid><br/><br/>var pivotdataSource =
{<br/>enableRowGrandTotal: false<br/>}; | property:
showRowGrandTotals<br/><br/><PivotViewComponent id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent><br/><br/>let
dataSourceSettings: IDataOptions = {<br/>showRowGrandTotals: false<br/>}; |
```

```
| Show/hide column grand totals | property: enableColumnGrandTotal<br/><br/><EJ.PivotGrid
id="PivotGrid" dataSource= {pivotdataSource}></EJ.PivotGrid><br/><br/>var pivotdataSource =
{<br/>enableColumnGrandTotal: false<br/>}; | property:
showColumnGrandTotals<br/><br/><PivotViewComponent id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent><br/><br/>let
dataSourceSettings: IDataOptions = {<br/>showColumnGrandTotals: false<br/>}; |
```

```
| Show/hide sub-totals | Not Applicable | property: showSubTotals<br/><br/><PivotViewComponent
id='PivotView' dataSourceSettings={dataSourceSettings}></PivotViewComponent><br/><br/>let
dataSourceSettings: IDataOptions = {<br/>showSubTotals: false<br/>}; |
```

```
| Show/hide row sub-totals | Not Applicable | property:
showRowSubTotals<br/><br/><PivotViewComponent id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent><br/><br/>let
dataSourceSettings: IDataOptions = {<br/>showRowSubTotals: false<br/>}; |
```

```
| Show/hide column sub-totals | Not Applicable | property:
showColumnSubTotals<br/><br/><PivotViewComponent id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent><br/><br/>let
dataSourceSettings: IDataOptions = {<br/>showColumnSubTotals: false<br/>}; |
```

```
| Show/hide sub-totals for specific field | property: showSubTotal<br/><br/><EJ.PivotGrid
id="PivotGrid" dataSource= {pivotdataSource}></EJ.PivotGrid><br/><br/>var pivotdataSource =
{<br/>rows: [{ name: 'company', showSubTotal: false }]<br/>}; | property:
showSubTotals<br/><br/><PivotViewComponent id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent><br/><br/>let
dataSourceSettings: IDataOptions = {<br/>rows: [{ name: 'company', showSubTotals: false
}]<br/>}; |
```

```
{% endraw %}
```

Drill operation

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

```
| Expand All | property: enableCollapseByDefault<br/><br/><EJ.PivotGrid id="PivotGrid"
enableCollapseByDefault= {true}></EJ.PivotGrid> | property:
expandAll<br/><br/><PivotViewComponent id='PivotView'
```

```

dataSourceSettings={dataSourceSettings}></PivotViewComponent><br><br>let
dataSourceSettings: IDataOptions = {<br>expandAll: false<br>};|

| Drill Up/Down | property: collapsedMembers<br><br><EJ.PivotGrid id="PivotGrid"
collapsedMembers= {collapsedMembers}></EJ.PivotGrid><br><br>var collapsedMembers =
{<br>Country: ["Canada", "France"]<br>};| property:
drilledMembers<br><br><PivotViewComponent id='PivotView'
dataSourceSettings={dataSourceSettings}></PivotViewComponent><br><br>let
dataSourceSettings: IDataOptions = {<br>drilledMembers: [{<br>name: 'Country',<br>items:
['France'] }]<br>};|

```

Field List

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Show/hide field list pop-up button on pivot table | Not Applicable | **property:**

```

showFieldList<br><br><PivotViewComponent id='PivotView'
showFieldList={true}></PivotViewComponent>|

```

| Defer update | **property:** enableDeferUpdate

<EJ.PivotGrid id="PivotGrid"
enableDeferUpdate= {true}></EJ.PivotGrid>| Not Applicable|

| Control initialization | **component:** PivotSchemaDesigner

<EJ.PivotSchemaDesigner
id="PivotSchemaDesigner"></EJ.PivotSchemaDesigner>| **component:**
PivotFieldListComponent

<PivotFieldListComponent id='PivotFieldList'>
</PivotFieldListComponent>|

| Render mode | Not Applicable | **property:** renderMode

<PivotFieldListComponent
id='PivotFieldList' renderMode= { 'Fixed' }></PivotFieldListComponent>|

| Show/hide calculated field button | Not Applicable | **property:**
allowCalculatedField

<PivotFieldListComponent id='PivotFieldList'
allowCalculatedField={true}></PivotFieldListComponent>|

| Show/hide value group button | Not Applicable | **property:**
showValuesButton

<PivotFieldListComponent id='PivotFieldList'
showValuesButton={true}></PivotFieldListComponent>|

Grouping Bar

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Show/hide Grouping bar | **property:** enableGroupingBar

<EJ.PivotGrid id="PivotGrid"
enableGroupingBar= {true}></EJ.PivotGrid>| **property:**
showGroupingBar

<PivotViewComponent id='PivotView'
showGroupingBar={true}></PivotViewComponent>|

| Grouping Bar Settings | Not Applicable | **property:**
groupingBarSettings

<PivotViewComponent id='PivotView'
groupingBarSettings={groupingBarSettings}></PivotViewComponent>

let

groupingBarSettings: GroupingBarSettings = {
showFilterIcon: false,
showSortIcon: true,
showRemoveIcon: false
};|

| Show/hide value group button | Not Applicable | **property:**

showValuesButton

<PivotFieldListComponent id='PivotFieldList'
showValuesButton={true}></PivotFieldListComponent>|

Filtering

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Filter settings | **property:** filterItems

<EJ.PivotGrid id="PivotGrid" dataSource={pivotdataSource}></EJ.PivotGrid>

var pivotdataSource = {
rows: [{
fieldName: "Country", fieldCaption: "Country",
filterItems : {
 filterType: ej.PivotAnalysis.FilterType.Exclude,
values: ["Canada", "France"] }
}}]; | **property:** filterSettings

<PivotViewComponent id="PivotGrid" dataSource={dataSource}></PivotViewComponent>

let dataSourceSettings: IDataOptions = {
filterSettings: [{
name: 'eyeColor',
type: 'Exclude',
items: ['blue'] }]
};|

Maximum node limit in member editor

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Max node limit in member editor | **property:** maxNodeLimitInMemberEditor

<EJ.PivotGrid id="PivotGrid" maxNodeLimitInMemberEditor= {100}></EJ.PivotGrid> | **property:** maxNodeLimitInMemberEditor

<PivotViewComponent id='PivotView' maxNodeLimitInMemberEditor={100}></PivotViewComponent>|

No Data Items

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Show/hide "no data" items | Not Applicable | **property:** showNoDataItems

<PivotViewComponent id='PivotView' dataSourceSettings={dataSourceSettings}></PivotViewComponent>

let dataSourceSettings: IDataOptions = {
rows: [{ name: 'company', showNoDataItems: true }]
};|

Excel-like filtering

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Label filtering | **property:** enableAdvancedFilter

<EJ.PivotGrid id="PivotGrid" enableAdvancedFilter={true} dataSource= {pivotdataSource}></EJ.PivotGrid>

var pivotdataSource = {
rows: [{
fieldName: "Country", fieldCaption: "Country",
advancedFilter : [{
labelFilterOperator: ej.olap.LabelFilterOptions.EndsWith,
values: ["es"] }]
}}]; | **property:** allowLabelFilter

<PivotViewComponent id="PivotGrid" allowLabelFilter= {true}></PivotViewComponent>|

```

dataSource= {dataSource}></PivotViewComponent><br><br>let dataSourceSettings:
IDataOptions = {<br>filterSettings: [{<br>name: 'product',<br>type: 'Label',<br>condition:
'Between',<br>value1: 'e', value2: 'v' }]<br>};|

|Value filtering|property: enableAdvancedFilter<br><br><EJ.PivotGrid id="PivotGrid"
enableAdvancedFilter={true} dataSource= {pivotdataSource}></EJ.PivotGrid><br><br>var
pivotdataSource = {<br>rows: [{<br>fieldName: "Country", fieldCaption:
"Country",<br>advancedFilter : [{<br>measure: "balance",<br>valueFilterOperator:
ej.olap.ValueFilterOptions.GreaterThan,<br>values: ["200"] }]<br>}}];|property:
allowValueFilter<br><br><PivotViewComponent id="PivotGrid" allowValueFilter= {true}
dataSource= {dataSource}></PivotViewComponent><br><br>let dataSourceSettings:
IDataOptions = {<br>filterSettings: [{<br>name: 'product',<br>measure: 'quantity',<br>type:
'Value',<br>condition: 'Between',<br>value1: '3250',<br>value2: '5000' }]<br>};|

```

Drill Through

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

```

|Show/hide drill though feature|property: enableDrillThrough<br><br><EJ.PivotGrid id="PivotGrid"
enableDrillThrough= {true}></EJ.PivotGrid>|property:
allowDrillThrough<br><br><PivotViewComponent id='PivotView'
allowDrillThrough={true}></PivotViewComponent>|

```

```

|Event Triggers when cell clicked in pivot table control|event: drillThrough<br><br><EJ.PivotGrid
id="PivotGrid" drillThrough= "onDrillThrough"></EJ.PivotGrid><br><br>function
onDrillThrough({})|event: drillThrough<br><br><PivotViewComponent id="PivotGrid"
drillThrough=
this.onDrillThrough.bind(this)></PivotViewComponent><br><br>onDrillThrough(): void {}|

```

Cell Editing

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

```

|Edit settings|Not Applicable|property: editSettings<br><br><PivotViewComponent
id='PivotView' editSettings={}></PivotViewComponent>|

```

```

|Show/hide cell editing feature|property: enableCellEditing<br><br><EJ.PivotGrid id="PivotGrid"
enableCellEditing= {true}></EJ.PivotGrid>|property:
allowEditing<br><br><PivotViewComponent id='PivotView'
editSettings={editSettings}></PivotViewComponent><br><br>let editSettings: CellEditSettings
= {<br>allowAdding: true, allowDeleting: true, allowEditing: true, mode: 'Normal'<br>};|

```

Hyperlink

```
{% raw %}
```

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Hyperlink settings | **property:** hyperlinkSettings

<EJ.PivotGrid id="PivotGrid"
hyperlinkSettings= {}></EJ.PivotGrid> | **property:**
hyperlinkSettings

<PivotViewComponent id='PivotView'
hyperlinkSettings={}></PivotViewComponent> |

| Show/hide hyperlink to all cells | Not Applicable | **property:**
showHyperlink

<PivotViewComponent id='PivotView'
hyperlinkSettings={hyperlinkSettings}></PivotViewComponent>

let hyperlinkSettings:
HyperLinkSettings = {
showHyperlink: 'true'
}; |

| Show/hide hyperlink to row headers | **property:** enableRowHeaderHyperlink

<EJ.PivotGrid
id="PivotGrid" hyperlinkSettings= {
enableRowHeaderHyperlink:
true
}></EJ.PivotGrid> | **property:** showRowHeaderHyperlink

<PivotViewComponent
id='PivotView' hyperlinkSettings={hyperlinkSettings}></PivotViewComponent>

let
hyperlinkSettings: HyperLinkSettings = {
showRowHeaderHyperlink: 'true'
}; |

| Show/hide hyperlink to column headers | **property:**
enableColumnHeaderHyperlink

<EJ.PivotGrid id="PivotGrid" hyperlinkSettings=
{
enableColumnHeaderHyperlink: true
}></EJ.PivotGrid> | **property:**
showColumnHeaderHyperlink

<PivotViewComponent id='PivotView'
hyperlinkSettings={hyperlinkSettings}></PivotViewComponent>

let hyperlinkSettings:
HyperLinkSettings = {
showColumnHeaderHyperlink: 'true'
}; |

| Show/hide hyperlink to value cells | **property:** enableValueCellHyperlink

<EJ.PivotGrid
id="PivotGrid" hyperlinkSettings= {
enableValueCellHyperlink:
true
}></EJ.PivotGrid> | **property:** showValueCellHyperlink

<PivotViewComponent
id='PivotView' hyperlinkSettings={hyperlinkSettings}></PivotViewComponent>

let
hyperlinkSettings: HyperLinkSettings = {
showValueCellHyperlink: 'true'
}; |

| Show/hide hyperlink to summary cells | **property:**
enableSummaryCellHyperlink

<EJ.PivotGrid id="PivotGrid" hyperlinkSettings=
{
enableSummaryCellHyperlink: true
}></EJ.PivotGrid> | **property:**
showSummaryCellHyperlink

<PivotViewComponent id='PivotView'
hyperlinkSettings={hyperlinkSettings}></PivotViewComponent>

let hyperlinkSettings:
HyperLinkSettings = {
showSummaryCellHyperlink: 'true'
}; |

| Show/hide hyperlink using specific conditions | Not Applicable | **property:**
conditionalSettings

<PivotViewComponent id='PivotView'
hyperlinkSettings={hyperlinkSettings}></PivotViewComponent>

let hyperlinkSettings:
HyperLinkSettings = {
conditionalSettings: [{
measure: 'Units Sold', conditions:
'Between', value1: 150, value2: 200
}] }></PivotViewComponent> |

| Show/hide hyperlink for row or column | Not Applicable | **property:**
headerText

<PivotViewComponent id='PivotView'
hyperlinkSettings={hyperlinkSettings}></PivotViewComponent>

let hyperlinkSettings:
HyperLinkSettings = {
headerText: 'FY 2015.Q1.Units Sold'
}; |

| Event Triggers when row headers clicked in pivot table | **event:**
rowHeaderHyperlinkClick

<EJ.PivotGrid id="PivotGrid" rowHeaderHyperlinkClick=


```

“onRowHeaderHyperlinkClick”></EJ.PivotGrid><br/><br/>function
onRowHeaderHyperlinkClick(){ }| event: hyperlinkCellClick<br/><br/><PivotViewComponent
id=“PivotGrid” hyperlinkCellClick=
this.onHyperlinkCellClick.bind(this)></PivotViewComponent><br/><br/>onHyperlinkCellClick():
void { }|

```

| Event Triggers when column headers clicked in pivot table | **event:**

```

columnHeaderHyperlinkClick<br/><br/><EJ.PivotGrid id=“PivotGrid” columnHeaderHyperlinkClick=
“onColumnHeaderHyperlinkClick”></EJ.PivotGrid><br/><br/>function
onColumnHeaderHyperlinkClick(){ }| event: hyperlinkCellClick<br/><br/><PivotViewComponent
id=“PivotGrid” hyperlinkCellClick=
this.onHyperlinkCellClick.bind(this)></PivotViewComponent><br/><br/>onHyperlinkCellClick():
void { }|

```

| Event Triggers when value cells clicked in pivot table | **event:**

```

valueCellHyperlinkClick<br/><br/><EJ.PivotGrid id=“PivotGrid” valueCellHyperlinkClick=
“onValueCellHyperlinkClick”></EJ.PivotGrid><br/><br/>function onValueCellHyperlinkClick(){
}| event: hyperlinkCellClick<br/><br/><PivotViewComponent id=“PivotGrid” hyperlinkCellClick=
this.onHyperlinkCellClick.bind(this)></PivotViewComponent><br/><br/>onHyperlinkCellClick():
void { }|

```

| Event Triggers when summary cells clicked in pivot table | **event:**

```

summaryCellHyperlinkClick<br/><br/><EJ.PivotGrid id=“PivotGrid” summaryCellHyperlinkClick=
“onSummaryCellHyperlinkClick”></EJ.PivotGrid><br/><br/>function
onSummaryCellHyperlinkClick(){ }| event: hyperlinkCellClick<br/><br/><PivotViewComponent
id=“PivotGrid” hyperlinkCellClick=
this.onHyperlinkCellClick.bind(this)></PivotViewComponent><br/><br/>onHyperlinkCellClick():
void { }|

```

```
{% endraw %}
```

Defer Layout Update

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Show/hide defer layout update | **property:** enableDeferUpdate

<EJ.PivotGrid
id=“PivotGrid” enableDeferUpdate= {true}></EJ.PivotGrid> | **property:**
allowDeferLayoutUpdate

<PivotViewComponent id=‘PivotView’
allowDeferLayoutUpdate={true}></PivotViewComponent> |

Sorting

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Enable/disable sorting | Not Applicable | **property:** enableSorting

<PivotViewComponent
id=“PivotGrid” dataSource= {dataSource}></PivotViewComponent>

let
dataSourceSettings: IDataOptions = {
enableSorting: false
};|

| Sort settings | **property:** sortOrder

<EJ.PivotGrid id="PivotGrid" dataSource={pivotdataSource}></EJ.PivotGrid>

var pivotdataSource = {
rows: [{
fieldName: "Country",
sortOrder : ej.PivotAnalysis.SortOrder.Descending}]
}; | **property:** sortSettings

<PivotViewComponent id="PivotGrid" dataSource={dataSource}></PivotViewComponent>

let dataSourceSettings: IDataOptions = {
sortSettings: [{
name: 'company',
order: 'Descending'}]
}; |

Value Sorting

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Enable/disable value sorting | Not Applicable | **property:** enableSorting

<PivotViewComponent id="PivotGrid" enableValueSorting={true}></PivotViewComponent> |

| Value sort settings | **property:** valueSortSettings

<EJ.PivotGrid id="PivotGrid" valueSortSettings={valueSortSettings}></EJ.PivotGrid>

var valueSortSettings = {
headerText: "Bike##Quantity",
headerDelimiters: "##",
sortOrder: ej.PivotAnalysis.SortOrder.Descending
}; | **property:** valueSortSettings

<PivotViewComponent id="PivotGrid" dataSource={dataSource}></PivotViewComponent>

let dataSourceSettings: IDataOptions = {
valueSortSettings: {
headerText: 'FY 2015##Sold Amount',
headerDelimiter: '##',
sortOrder: 'Descending' }
}; |

Calculated Field

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Show/hide calculated field | Not Applicable | **property:** allowCalculatedField

<PivotViewComponent id="PivotGrid" allowCalculatedField={true}></PivotViewComponent> |

| Calculated field settings | **property:** values

<EJ.PivotGrid id="PivotGrid" dataSource={pivotdataSource}></EJ.PivotGrid>

var pivotdataSource = {
values: [
{ fieldName: "Amount", fieldCaption: "Amount"},
{ fieldName: "Price",
fieldCaption: "Price",
isCalculatedField: true,
formula: "Amount*15" }]
 }; | **property:** calculatedFieldSettings

<PivotViewComponent id="PivotGrid" dataSource={dataSource}></PivotViewComponent>

let dataSourceSettings: IDataOptions = {
values: [{
 name: 'Total', type: 'CalculatedField' }],
calculatedFieldSettings: [{
name: 'Total',
formula: "Sum(Amount)"+"Sum(Sold)" }]
}; |

Paging

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Paging | **property:** enablePaging

<EJ.PivotGrid id="PivotGrid" enablePaging={true}></EJ.PivotGrid> | Not Applicable |

| Virtual scrolling | **property:** enableVirtualScrolling

<EJ.PivotGrid id="PivotGrid" enableVirtualScrolling= {true}></EJ.PivotGrid> | **property:** enableVirtualization

<PivotViewComponent id="PivotGrid" enableVirtualization= {true}></PivotViewComponent> |

Conditional Formatting

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Show/hide conditional formatting | **property:** enableConditionalFormatting

<EJ.PivotGrid id="PivotGrid" enableConditionalFormatting= {true}></EJ.PivotGrid> | **property:** allowConditionalFormatting

<PivotViewComponent id="PivotGrid" allowConditionalFormatting= {true}></PivotViewComponent> |

| Conditional formatting settings | **property:** conditionalFormatSettings

<EJ.PivotGrid id="PivotGrid" conditionalFormatSettings= {conditionalFormatSettings}></EJ.PivotGrid>

var conditionalFormatSettings = [{
name: "Format2",
style: {
"color": "#000000",
"backgroundcolor": "#0000FF",
"bordercolor": "#000000",
"borderstyle": "Dashed",
"borderwidth": "5",
"fontsize": "12",
"fontstyle": "Algerian" },
condition: ej.PivotGrid.ConditionalOptions.LessThan,
value: "200",
measures: "Amount,Quantity"}]; | **property:** conditionalFormatSettings

<PivotViewComponent id="PivotGrid" dataSource= {dataSource}></PivotViewComponent>

let dataSourceSettings: IDataOptions = {
conditionalFormatSettings: [{
measure: 'In Stock',
value1: 5000,
conditions: 'LessThan',
style: {
backgroundcolor: '#80cbc4',
 color: 'black',
 fontFamily: 'Tahoma',
fontSize: '12px' } }]
}; |

Exporting

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Excel Export | Not Applicable | **property:** allowExcelExport

<PivotViewComponent id="PivotGrid" allowExcelExport= {true}></PivotViewComponent> |

| Pdf Export | Not Applicable | **property:** allowPdfExport

<PivotViewComponent id="PivotGrid" allowPdfExport= {true}></PivotViewComponent> |

Grid Customization

{% raw %}

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Set width for pivot table | Not Applicable | **property:** width

<PivotViewComponent id="PivotGrid" width= {800}></PivotViewComponent> |

| Set height for pivot table | Not Applicable | **property:** height

<PivotViewComponent id="PivotGrid" height= {400}></PivotViewComponent> |

|Set row height for pivot table| Not Applicable | **property:** `rowHeight`
`<PivotViewComponent id="PivotGrid" gridSettings= {{rowHeight: 60}}></PivotViewComponent>` |

|Set column width for pivot table| Not Applicable | **property:** `columnWidth`
`<PivotViewComponent id="PivotGrid" gridSettings= {{columnWidth: 120}}></PivotViewComponent>` |

|Drag and drop column headers in pivot table| Not Applicable | **property:** `allowReordering`
`<PivotViewComponent id="PivotGrid" gridSettings= {{allowReordering: true}}></PivotViewComponent>` |

|Resizing the column headers in pivot table| **property:** `enableColumnResizing`
`<EJ.PivotGrid id="PivotGrid" enableColumnResizing= {true}></EJ.PivotGrid>` | **property:** `allowResizing`
`<PivotViewComponent id="PivotGrid" gridSettings= {{allowResizing: true}}></PivotViewComponent>` |

|Wrap the cell content in pivot table| Not Applicable | **property:** `allowTextWrap`
`<PivotViewComponent id="PivotGrid" gridSettings= {{allowTextWrap: true}}></PivotViewComponent>` |

|Display cell border in pivot table| Not Applicable | **property:** `gridLines`
`<PivotViewComponent id="PivotGrid" gridSettings= {{gridLines:'Vertical'}}></PivotViewComponent>` |

|Cell selection| **property:** `enableCellSelection`
`<EJ.PivotGrid id="PivotGrid" enableCellSelection= {true}></EJ.PivotGrid>` | **property:** `allowSelection`
`<PivotViewComponent id="PivotGrid" gridSettings= {gridSettings}></PivotViewComponent>`
`let gridSettings: GridSettings = {allowSelection: true,selectionSettings: {cellSelectionMode: 'Box', type: 'Multiple', mode: 'Cell' }</>};` |

|Display overflow cell content in pivot table| Not Applicable | **property:** `clipMode`
`<PivotViewComponent id="PivotGrid" gridSettings= {{clipMode: 'Clip'}}></PivotViewComponent>` |

|Cell Editing| **property:** `enableCellEditing`
`<EJ.PivotGrid id="PivotGrid" enableCellEditing= {true}></EJ.PivotGrid>` | Not Applicable |

|Cell double click| **property:** `enableCellDoubleClick`
`<EJ.PivotGrid id="PivotGrid" enableCellDoubleClick= {true}></EJ.PivotGrid>` | Not Applicable |

|Cell context| **property:** `enableCellContext`
`<EJ.PivotGrid id="PivotGrid" enableCellContext= {true}></EJ.PivotGrid>` | Not Applicable |

{% endraw %}

Accessibility

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Localization | **property:** locale

<EJ.PivotGrid id="PivotGrid" locale= { 'es-ES' }></EJ.PivotGrid> | **property:** locale

<PivotViewComponent id="PivotGrid" locale= { 'es-ES' }></PivotViewComponent> |

| Right to left | **property:** enableRTL

<EJ.PivotGrid id="PivotGrid" enableRTL= { true }></EJ.PivotGrid> | **property:** enableRtl

<PivotViewComponent id="PivotGrid" enableRTL= { true }></PivotViewComponent> |

Common

{% raw %}

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

| --- | --- | --- |

| Adding custom class to wrapper element | **property:** cssClass

<EJ.PivotGrid id="PivotGrid" cssClass= { "custom-class" }></EJ.PivotGrid> | **property:** cssClass

<PivotViewComponent id="PivotGrid" cssClass= { 'custom-class' }></PivotViewComponent> |

| Keeping the model values in cookies | Not Applicable | **property:** enablePersistence

<PivotViewComponent id="PivotGrid" enablePersistence= { true }></PivotViewComponent> |

| Event triggers when control initializing | **event:** load

<EJ.PivotGrid id="PivotGrid" load= "onLoad"></EJ.PivotGrid>

function onLoad() { } | **event:** load

<PivotViewComponent id="PivotGrid" load= this.onLoad.bind(this)></PivotViewComponent>

onLoad(): void { } |

| Event Triggers before the pivot engine starts | **event:** beforePivotEnginePopulate

<EJ.PivotGrid id="PivotGrid" beforePivotEnginePopulate= "onBeforePivotEnginePopulate"></EJ.PivotGrid>

function onBeforePivotEnginePopulate() { } | **event:** enginePopulating

<PivotViewComponent id="PivotGrid" enginePopulating= this.onEnginePopulating.bind(this)></PivotViewComponent>

onEnginePopulating(): void { } |

| Event Triggers after the pivot engine populated | **event:** afterPivotEnginePopulate

<EJ.PivotGrid id="PivotGrid" afterPivotEnginePopulate= "onAfterPivotEnginePopulate"></EJ.PivotGrid>

function onAfterPivotEnginePopulate() { } | **event:** enginePopulated

<PivotViewComponent id="PivotGrid" enginePopulated= this.onEnginePopulated.bind(this)></PivotViewComponent>

onEnginePopulated(): void { } |

| Event Triggers after the control populated with data source | **event:** renderSuccess

<EJ.PivotGrid id="PivotGrid" renderSuccess= "onRenderSuccess"></EJ.PivotGrid>

function onRenderSuccess() { } | **event:** dataBound

<PivotViewComponent id="PivotGrid" dataBound= this.onDataBound.bind(this)></PivotViewComponent>

onDataBound(): void { } |

```
| Event Triggers after the control created | Not Applicable | event:
created<br/><br/><PivotViewComponent id="PivotGrid" created=
this.created.bind(this)></PivotViewComponent><br/><br/>created(): void { }|
```

```
| Event Triggers when destroy the control | Not Applicable | event:
destroyed<br/><br/><PivotViewComponent id="PivotGrid" destroyed=
this.destroyed.bind(this)></PivotViewComponent><br/><br/>destroyed(): void { }|
```

```
| Event Triggers the cell clicked in pivot table | event: cellClick<br/><br/><EJ.PivotGrid id="PivotGrid"
cellClick= "onCellClick"></EJ.PivotGrid><br/><br/>function onCellClick(){ } | event:
cellClick<br/><br/><PivotViewComponent id="PivotGrid" cellClick=
this.onCellClick.bind(this)></PivotViewComponent><br/><br/>onCellClick(): void { }|
```

```
{% endraw %}
```

How To

```
<!-- markdownlint-disable MD009 -->
```

Switching older themes style in React Pivotview component

From Volume 1, 2020 onwards Syncfusion has revised the theming and layout of the Pivot Table. So, to inherit the older theme style and layout please do the necessary changes in CSS and pivot table height.

CSS Selectors

In current theme, the cells can be differentiated by their background colors. To avoid it, you need to override its background colors via simple CSS coding within your application. The below CSS selectors allow to achieve the same for material, fabric, bootstrap and bootstrap v4 themes.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<!-- Codes here... -->
<style>
.e-pivotview .e-rowsheader,
.e-pivotview .e-columnsheader,
.e-pivotview .e-gtot,
.e-pivotview .e-content,
.e-pivotview .e-gridheader,
.e-pivotview .e-headercell {
background-color:#fff !important;
}
</style>
</head>
```

```
<body>
</body>
</html>
`
```

Meanwhile for high contrast theme, we need to set the following CSS.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<!-- Codes here... -->
<style>
.e-pivotview .e-rowsheader,
.e-pivotview .e-columnsheader,
.e-pivotview .e-gtot,
.e-pivotview .e-content,
.e-pivotview .e-gridheader,
.e-pivotview .e-headercell {
background-color:#000 !important;
}
</style>
</head>
<body>
</body>
</html>
`
```

Adjusting Row Height

In current theme, to make the component compact we have reduced the height of each pivot table rows. But user can reset the height of the pivot table using the [rowHeight](#) property in [gridSettings](#). In older theme, the property was set to 36 pixels for desktop layout and 48 pixels for mobile layout. So reset the [rowHeight](#) accordingly to visualize the older theme style.

In the below code sample, we replicate the older theme style.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
```

```
function App() {
  let gridSettings = {
    rowHeight: 36
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
function App() {
  let gridSettings: GridSettings = {
    rowHeight: 36
  } as GridSettings;
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} gridSettings={gridSettings}
dataSourceSettings={dataSourceSettings} ></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Customize number date and time values in React Pivotview component

You can format the number, date, and time values for each field using `formatSettings` option under `dataSourceSettings`. It can be configured through code behind, during initial rendering.

Number formatting

For numbers, the formatting settings required to apply through code behind are:

- `name`: It allows to set the field name.
- `format`: It allows to set the format of the respective field.

Also, you can customize the applied number format by setting the `NumberFormatOptions` options in `formatSettings` itself.

INDEX.JSX

```
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C2', useGrouping: false,
minimumSignificantDigits: 1, maximumSignificantDigits: 3 }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
```



```

    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C2', useGrouping: false,
minimumSignificantDigits: 1, maximumSignificantDigits: 3 }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Date and Time formatting

For date and time, the formatting settings required to apply through code behind are:

- **name**: It allows to set the field name.
- **format**: It allows to set the format of the respective field.
- **type**: It allows to set the type of format to be used for the respective field.

Also, you can customize the applied date format by setting [DateFormatOptions](#) options in **formatSettings** itself.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    dataSource: pivotData,
    expandAll: false,
    allowLabelFilter: true,
    formatSettings: [{ name: 'Year', format: 'dd/MM/yyyy-hh:mm', type:
'date' }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    filters: []
  };
  let pivotObj;

```

```

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataset[],
        expandAll: false,
        allowLabelFilter: true,
        formatSettings: [{ name: 'Year', format: 'dd/MM/yyyy-hh:mm', type:
'date' }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        filters: []
    }
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Limitations of date formatting

As per Firefox and Edge browsers standards, most of the date and time formats used in data source aren't supported. For example: Apr-2000, Apr-01-2000, 01-03-2000, 2000-Apr-01 etc... are not supported. Meanwhile [ISO formats](#) will be supported across all browsers.

Customize the icons for pivot table in React Pivotview component

You can customize the pivot button icons in the pivot table by overriding the class **.pivot-button** with a custom property content as mentioned below.

```

`ts
PivotView_PivotFieldList.e-icons.e-toggle-field-list::before {
content: '\e337';
}
`
`ts

```

PivotView_PivotFieldList.e - icons.e - toggle - field - list;
before;

```
{
content: '\e337';
}
,
```

In the below sample, pivot table is rendered with a customized pivot button icons.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C2', useGrouping: false,
minimumSignificantDigits: 1, maximumSignificantDigits: 3 }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}/></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
```

```

    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France'] }],
    formatSettings: [{ name: 'Amount', format: 'C2', useGrouping: false,
minimumSignificantDigits: 1, maximumSignificantDigits: 3 }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
  }
  let pivotObj: PivotViewComponent;

  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={ [FieldList]} /></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Configure data grid options on editing mode in React Pivotview component

You can access the data grid options such as sort, group, filter, etc on editing mode using **beginDrillThrough** event in the pivot table. The event fires on every value cell click and provides the data grid information before it displays.

To access the data grid options, you need to inject module for the provided options in data grid itself.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { Grid, Sort, Filter, Group } from '@syncfusion/ej2-grids';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let editSettings = {
    allowAdding: true, allowDeleting: true, allowEditing: true, mode:
'Normal'
  };
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France'] }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
  };
  function beginDrillThrough(args) {
    if (args.gridObj) {
      Grid.Inject(Sort, Filter, Group);
      let gridObj = args.gridObj;
      gridObj.allowGrouping = true;
      gridObj.allowSorting = true;
    }
  }
}

```

```

        gridObj.allowFiltering = true;
        gridObj.filterSettings = { type: 'CheckBox' };
    }
}
let pivotObj;
return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
editSettings={editSettings}
beginDrillThrough={beginDrillThrough.bind(this)}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, CellEditSettings, IDataset, Inject,
PivotViewComponent, BeginDrillThroughEventArgs } from '@syncfusion/ej2-
react-pivotview';
import { Grid, Sort, Filter, Group } from '@syncfusion/ej2-grids';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
function App() {
    let editSettings: CellEditSettings = {
        allowAdding: true, allowDeleting: true, allowEditing: true, mode:
'Normal'
    } as CellEditSettings
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    }
    function beginDrillThrough(args: BeginDrillThroughEventArgs): void {
        if (args.gridObj) {
            Grid.Inject(Sort, Filter, Group);
            let gridObj: Grid = args.gridObj;
            gridObj.allowGrouping = true;
            gridObj.allowSorting = true;
            gridObj.allowFiltering = true;
            gridObj.filterSettings = { type: 'CheckBox' };
        }
    }
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
editSettings={editSettings}
beginDrillThrough={beginDrillThrough.bind(this)}></PivotViewComponent>);

```

```
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

Refresh the field list in React Pivotview component

You can refresh pivot table and field list with new data source dynamically.

INDEX.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { PivotViewComponent, FieldList, Inject } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }],
        dataSource: [
            { 'Sold': 31, 'Amount': 52824, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q1' },
            { 'Sold': 51, 'Amount': 86904, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q2' },
            { 'Sold': 90, 'Amount': 153360, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q3' },
            { 'Sold': 25, 'Amount': 42600, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q4' },
            { 'Sold': 27, 'Amount': 46008, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2016', 'Quarter': 'Q1' }
        ],
        expandAll: false,
        filters: [],
        rows: [{ name: 'Country' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }]
    };
    let pivotObj;
    return (<div><div className="col-md-9"> <PivotViewComponent ref={d => pivotObj = d} id="PivotView" height={350}
    dataSourceSettings={dataSourceSettings} showFieldList={true}><Inject
    services={[FieldList]}></PivotViewComponent></div><div className="col-lg-3
    property-section"><ButtonComponent cssClass="e-primary"
    onClick={btnClick.bind(this)}>Refresh</ButtonComponent></div></div>);
    function btnClick() {
        pivotObj.engineModule.fieldList = {};
        pivotObj.dataSourceSettings.dataSource = [
            { 'Sold': 31, 'Amount': 52824, 'Country': 'France', 'Year': 'FY 2015' },
            { 'Sold': 51, 'Amount': 86904, 'Country': 'France', 'Year': 'FY 2015' },
            { 'Sold': 90, 'Amount': 153360, 'Country': 'France', 'Year': 'FY 2015' },
            { 'Sold': 25, 'Amount': 42600, 'Country': 'France', 'Year': 'FY 2015' },
            { 'Sold': 27, 'Amount': 46008, 'Country': 'France', 'Year': 'FY 2016' }
        ];
    }
}
```

```

}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { IDataOptions, IDataset, PivotViewComponent, FieldList, Inject }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }],
        dataSource: [
            { 'Sold': 31, 'Amount': 52824, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q1' },
            { 'Sold': 51, 'Amount': 86904, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q2' },
            { 'Sold': 90, 'Amount': 153360, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q3' },
            { 'Sold': 25, 'Amount': 42600, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q4' },
            { 'Sold': 27, 'Amount': 46008, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2016', 'Quarter': 'Q1' } ] as IDataset[],
        expandAll: false,
        filters: [],
        rows: [{ name: 'Country' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }]
    };
    let pivotObj: PivotViewComponent;

    return (<div><div className="col-md-9"> <PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={350}
dataSourceSettings={dataSourceSettings} showFieldList={true}><Inject
services={[FieldList]} /></PivotViewComponent></div><div className='col-lg-3
property-section'><ButtonComponent cssClass='e-primary'
onClick={btnClick.bind(this)}>Refresh</ButtonComponent></div></div>);
    function btnClick(): void {
        pivotObj.engineModule.fieldList = {};
        pivotObj.dataSourceSettings.dataSource = [
            { 'Sold': 31, 'Amount': 52824, 'Country': 'France', 'Year': 'FY
2015' },
            { 'Sold': 51, 'Amount': 86904, 'Country': 'France', 'Year': 'FY
2015' },
            { 'Sold': 90, 'Amount': 153360, 'Country': 'France', 'Year': 'FY
2015' },
            { 'Sold': 25, 'Amount': 42600, 'Country': 'France', 'Year': 'FY
2015' },
            { 'Sold': 27, 'Amount': 46008, 'Country': 'France', 'Year': 'FY
2016' } ];
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Hide empty headers in React Pivotview component

If the raw data for a particular field is not defined, it will be shown as 'Undefined' in the pivot table headers. You can hide those headers by setting the [showHeaderWhenEmpty](#) property to **false** in the pivot table.

For example, if the raw data for the field 'Country' is defined as “United Kingdom” and “State” is not defined means, it will be shown as “United Kingdom >> Undefined” in the header section. Here, you can hide those 'Undefined' header using the [showHeaderWhenEmpty](#) property.

By default, this property is set as **true**.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotNullData } from './datasource';
function App() {
    let dataSourceSettings = {
        dataSource: pivotNullData,
        expandAll: false,
        rows: [{ name: 'Country' }, { name: 'State' }],
        columns: [{ name: 'Product', showNoDataItems: true }, { name: 'Date'
    }],
        values: [{ name: 'Amount' }, { name: 'Quantity' }],
        showHeaderWhenEmpty: false
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotNullData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotNullData as IDataset[],
        expandAll: false,
        rows: [{ name: 'Country' }, { name: 'State' }],
        columns: [{ name: 'Product', showNoDataItems: true }, { name: 'Date'
    }],
        values: [{ name: 'Amount' }, { name: 'Quantity' }],
        showHeaderWhenEmpty: false
    }
}
```



```

let pivotObj: PivotViewComponent;

return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
showFieldList={true}><Inject services={[FieldList]}
/></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Customizing loading indicator in React Pivotview component

You can customize the appearance of the loading indicator in the pivot table by using the [spinnerTemplate](#) property. This property accepts an HTML string which can be used for appearance customization.

You can also disable the loading indicator by setting [spinnerTemplate](#) to empty string.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { DataManager, WebApiAdaptor, Query } from '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

function App() {
    let dataSourceSettings;
    let pivotObj;
    let dataSource = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    }).executeQuery(new Query().take(8)).then((e) => {
        pivotObj.dataSourceSettings = {
            dataSource: e.result,
            expandAll: true,
            filters: [],
            columns: [{ name: 'ProductName', caption: 'Product Name' }],
            rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
            formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
            values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption:
'Unit Price' }]
        };
    });

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} spinnerTemplate={'<i class="fa fa-cog fa-spin fa-3x fa-
fw"></i>'} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';

```

```

import { DataManager, WebApiAdaptor, Query, ReturnOption } from
 '@syncfusion/ej2-data';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let dataSourceSettings: IDataOptions;
  let pivotObj: PivotViewComponent;
  let dataSource: Promise<void> = new DataManager({
    url: 'https://bi.syncfusion.com/northwindservice/api/orders',
    adaptor: new WebApiAdaptor,
    crossDomain: true
  }).executeQuery(new Query().take(8)).then((e: ReturnOption) => {
    pivotObj.dataSourceSettings = {
      dataSource: e.result as IDataset[],
      expandAll: true,
      filters: [],
      columns: [{ name: 'ProductName', caption: 'Product Name' }],
      rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
      formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
      values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption: 'Unit
Price' }]
    }
  });
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} spinnerTemplate={<i class="fa fa-cog fa-spin fa-3x fa-
fw"></i>} dataSourceSettings={dataSourceSettings}></PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Changing the pivotview component minimum height in React Pivotview component

The **minHeight** property allows you to change the minimum height for the pivot table control. For the pivot table control, the default minimum height is **300px**.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  };
  let pivotObj;
  function dataBound() {

```

```

        if (pivotObj) {
            pivotObj.minHeight = 200;
        }
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={200} dataSourceSettings={dataSourceSettings}
dataBound={dataBound.bind(this)}>
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let pivotObj: PivotViewComponent;
    function dataBound(): void {
        if(pivotObj) {
            pivotObj.minHeight = 200;
        }
    }

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={200} dataSourceSettings={dataSourceSettings}
dataBound={dataBound.bind(this)}>
    </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Chart based on pivot table selection in React Pivotview component

The cell selection support is enabled using the [allowSelection](#) property and its type and mode are configured using the [selectionSettings](#) property. The [cellSelected](#) event gets fired on every selection operation performed in the pivot table. This event returns the selected cell informations, like row header name, column header name, measure name, and value. Based on this information, the [chart](#) control will be plotted.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Inject, PivotViewComponent, FieldList } from '@syncfusion/ej2-react-pivotview';
import { Chart, Category, Legend, Tooltip, ColumnSeries, LineSeries } from '@syncfusion/ej2-charts';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' } ]
    };
    let gridSettings = {
        columnWidth: 120,
        allowSelection: true,
        selectionSettings: {
            mode: 'Cell',
            type: 'Multiple',
            cellSelectionMode: 'Box',
        }
    };
    let onInit = true;
    let measureList = {};
    let chart;
    let selectedCells;
    let chartSeries;
    let pivotObj;
    function frameChartSeries() {
        let columnGroupObject = {};
        for (let cell of selectedCells) {
            if (cell.measure !== '') {
                let columnSeries =
                (pivotObj.dataSourceSettings.values.length > 1 && measureList[cell.measure])
                ?
                    (cell.columnHeaders.toString() + ' ~ ' +
                    measureList[cell.measure]) : cell.columnHeaders.toString();
                if (columnGroupObject[columnSeries]) {
                    columnGroupObject[columnSeries].push({ x:
                    cell.rowHeaders == '' ? 'Grand Total' : cell.rowHeaders.toString(), y:
                    Number(cell.value) });
                }
                else {
                    columnGroupObject[columnSeries] = [{ x: cell.rowHeaders
                    == '' ? 'Grand Total' : cell.rowHeaders.toString(), y: Number(cell.value)
                    }];
                }
            }
        }
    }
}

```

```

    }
    let columnKeys = Object.keys(columnGroupObject);
    let chartSeries = [];
    for (let key of columnKeys) {
        chartSeries.push({
            dataSource: columnGroupObject[key],
            xName: 'x',
            yName: 'y',
            type: 'Column',
            name: key
        });
    }
    return chartSeries;
}
function chartUpdate() {
    if (onInit) {
        onInit = false;
        Chart.Inject(ColumnSeries, LineSeries, Legend, Tooltip,
Category);
        chart = new Chart({
            title: 'Sales Analysis',
            legendSettings: {
                visible: true
            },
            tooltip: {
                enable: true
            },
            primaryYAxis: {
                title: pivotObj.dataSourceSettings.values.map(function
(args) { return args.caption || args.name; }).join(' ~ '),
            },
            primaryXAxis: {
                valueType: 'Category',
                title: pivotObj.dataSourceSettings.rows.map(function
(args) { return args.caption || args.name; }).join(' ~ '),
                labelIntersectAction: 'Rotate45'
            },
            series: chartSeries,
        }, '#Chart');
    }
    else {
        chart.series = chartSeries;
        chart.primaryXAxis.title =
pivotObj.dataSourceSettings.rows.map(function (args) { return args.caption
|| args.name; }).join(' ~ ');
        chart.primaryYAxis.title =
pivotObj.dataSourceSettings.values.map(function (args) { return args.caption
|| args.name; }).join(' ~ ');
        chart.refresh();
    }
}
function dataBound() {
    if (onInit) {
        for (let value of pivotObj.dataSourceSettings.values) {
            measureList[value.name] = value.caption || value.name;
        }
    }
}

```

```

        pivotObj.grid.selectionModule.selectCellsByRange({ cellIndex: 1,
rowIndex: 1 }, { cellIndex: 3, rowIndex: 3 });
    }
}
function cellSelected(args) {
    selectedCells = args.selectedCellsInfo;
    if (selectedCells && selectedCells.length > 0) {
        chartSeries = frameChartSeries();
        chartUpdate();
    }
}
return (<div className="control-section"><PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={300}
dataSourceSettings={dataSourceSettings} showFieldList={true}
gridSettings={gridSettings}><Inject
services={[FieldList]}></PivotViewComponent><br /><div
id="Chart"></div></div></div>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { IDataOptions, IDataset, Inject, PivotViewComponent, FieldList }
from '@syncfusion/ej2-react-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Chart, Category, Legend, Tooltip, ColumnSeries, LineSeries,
SeriesModel } from '@syncfusion/ej2-charts';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        dataSource: pivotData as IDataset[],
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
    };
    let gridSettings: GridSettings = {
        columnWidth: 120,
        allowSelection: true,
        selectionSettings: {
            mode: 'Cell',
            type: 'Multiple',
            cellSelectionMode: 'Box',
        }
    };
    let onInit: boolean = true;

```

```

let measureList: { [key: string]: string } = {};
let chart: Chart;
let selectedCells: CellSelectedObject[];
let chartSeries: SeriesModel[];
let pivotObj: PivotViewComponent;
function frameChartSeries(): SeriesModel[] {
    let columnGroupObject: { [key: string]: { x: string, y: number }[] }
= {};
    for (let cell of selectedCells) {
        if (cell.measure !== '') {
            let columnSeries = (pivotObj.dataSourceSettings.values.length >
1 && measureList[cell.measure]) ?
            (cell.columnHeaders.toString() + ' ~ ' +
measureList[cell.measure]) : cell.columnHeaders.toString();
            if (columnGroupObject[columnSeries]) {
                columnGroupObject[columnSeries].push({ x: cell.rowHeaders == ''
? 'Grand Total' : cell.rowHeaders.toString(), y: Number(cell.value) });
            } else {
                columnGroupObject[columnSeries] = [{ x: cell.rowHeaders == '' ?
'Grand Total' : cell.rowHeaders.toString(), y: Number(cell.value) }];
            }
        }
    }
    let columnKeys: string[] = Object.keys(columnGroupObject);
    let chartSeries: SeriesModel[] = [];
    for (let key of columnKeys) {
        chartSeries.push({
            dataSource: columnGroupObject[key],
            xName: 'x',
            yName: 'y',
            type: 'Column',
            name: key
        });
    }
    return chartSeries;
}
function chartUpdate(): void {
    if (onInit) {
        onInit = false;
        Chart.Inject(ColumnSeries, LineSeries, Legend, Tooltip,
Category);
        chart = new Chart({
            title: 'Sales Analysis',
            legendSettings: {
                visible: true
            },
            tooltip: {
                enable: true
            },
            primaryYAxis: {
                title: pivotObj.dataSourceSettings.values.map(function
(args) { return args.caption || args.name }).join(' ~ '),
            },
            primaryXAxis: {
                valueType: 'Category',
                title: pivotObj.dataSourceSettings.rows.map(function
(args) { return args.caption || args.name }).join(' ~ '),

```

```

        labelIntersectAction: 'Rotate45'
      },
      series: chartSeries,
    }, '#Chart');
  } else {
    chart.series = chartSeries;
    chart.primaryXAxis.title =
pivotObj.dataSourceSettings.rows.map(function (args) { return args.caption
|| args.name }).join(' ~ ');
    chart.primaryYAxis.title =
pivotObj.dataSourceSettings.values.map(function (args) { return args.caption
|| args.name }).join(' ~ ');
    chart.refresh();
  }
}
function dataBound(): void {
  if(onInit) {
    for (let value of pivotObj.dataSourceSettings.values) {
      measureList[value.name] = value.caption || value.name;
    }
    pivotObj.grid.selectionModule.selectCellsByRange(
      { cellIndex: 1, rowIndex: 1 },
      { cellIndex: 3, rowIndex: 3 }
    );
  }
}
function cellSelected(args: PivotCellSelectedEventArgs): void {
  selectedCells = args.selectedCellsInfo;
  if (selectedCells && selectedCells.length > 0) {
    chartSeries = frameChartSeries();
    chartUpdate();
  }
}

return (<div className="control-section"><PivotViewComponent ref={d =>
pivotObj = d} id='PivotView' height={300}
dataSourceSettings={dataSourceSettings} showFieldList={true}
gridSettings={gridSettings}><Inject
services={[FieldList]}/></PivotViewComponent><br/><div
id="Chart"></div></div>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Drill through grid cell edit type in React Pivotview component

Using the [drillThrough](#) event in the pivot table, you can define the edit type of a particular column in the grid present inside the drill-through dialog. To do so, check the column name in the [drillThrough](#) event and then specify the edit type of that column using the [gridColumns.editType](#) event argument.

The [gridColumns.editType](#) property must be set based on the column's data type. For example, the string data type will not be applicable for the numeric text box edit type.

- [NumericTextBox](#) control for integer, double, and decimal data types.
- [TextBox](#) control for string data type.

- [DropDownList](#) control to show all unique values related to that field.
- [CheckBox](#) control for boolean data type.
- [DatePicker](#) control for date data type.
- [DateTimePicker](#) control for date time data type.

In the below example, the data type of the **Country** column is set to **DropDownList**.

INDEX.JSX

```
{% raw %}
import { DrillThrough, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        dataSource: pivotData,
        expandAll: false,
        filters: [],
        drilledMembers: [{ name: 'Country', items: ['France' ]}],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
    };
    let pivotObj;
    function drillThrough(args) {
        for (var i = 0; i < args.gridColumns.length; i++) {
            if (args.gridColumns[i].field === 'Country') {
                args.gridColumns[i].editType = 'dropdownedit';
                //args.gridColumns[i].editType = 'numericedit';
                //args.gridColumns[i].editType = 'textedit';
                //args.gridColumns[i].editType = 'booleanedit';
                //args.gridColumns[i].editType = 'datepickeredit';
                //args.gridColumns[i].editType = 'datetimepickeredit';
            }
        }
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView' height={350} dataSourceSettings={dataSourceSettings} allowDrillThrough={true} editSettings={{ allowEditing: true, allowAdding: true, allowDeleting: true, mode: 'Normal' }} drillThrough={drillThrough.bind(this)}><Inject services={[DrillThrough]}></PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
```

```

import { DrillThrough, IDataOptions, IDataset, Inject, PivotViewComponent }
  from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataset[],
    expandAll: false,
    filters: [],
    drilledMembers: [{ name: 'Country', items: ['France' ]}],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  function drillThrough(args: BeginDrillThroughEventArgs): void {
    for (var i = 0; i < args.gridColumns.length; i++) {
      if (args.gridColumns[i].field === 'Country') {
        args.gridColumns[i].editType = 'dropdownedit';
        //args.gridColumns[i].editType = 'numericedit';
        //args.gridColumns[i].editType = 'textedit';
        //args.gridColumns[i].editType = 'booleanedit';
        //args.gridColumns[i].editType = 'datepickeredit';
        //args.gridColumns[i].editType = 'datetimepickeredit';
      }
    }
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
allowDrillThrough={true}editSettings={{ allowEditing: true, allowAdding:
true, allowDeleting: true, mode: 'Normal'}}
drillThrough={drillThrough.bind(this)} ><Inject services={[DrillThrough]}/>
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

Show field list when pivot table empty in React Pivotview component

When there are no fields in a pivot table's row, column, value, and filter axes, a field list can still be displayed. To do so, use the [dataBound](#) event and call the `onShowFieldList` method as shown below.

INDEX.JSX

```

import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {

```

```

        dataSource: pivotData,
    };
    let pivotObj;
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
dataBound={dataBound.bind(this)} showFieldList={true}><Inject
services={[FieldList]} /></PivotViewComponent>);
    function dataBound() {
        if (pivotObj && pivotObj.dataSourceSettings.values.length === 0) {
            pivotObj.pivotFieldListModule.dialogRenderer.onShowFieldList();
        }
    }
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataset[],
    }
    let pivotObj: PivotViewComponent;

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings}
dataBound={dataBound.bind(this)} showFieldList={true}><Inject
services={[FieldList]} /></PivotViewComponent>);
    function dataBound(): void {
        if (pivotObj && pivotObj.dataSourceSettings.values.length === 0) {
            (pivotObj.pivotFieldListModule.dialogRenderer as
any).onShowFieldList();
        }
    }
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Apply custom style to pivot cells in React Pivotview component

The [queryCellInfo](#) event in [gridSettings](#) can be used to apply custom style to row and value cells, and the [headerCellInfo](#) event in [gridSettings](#) can be used to apply custom styles to column cells.

In the following example, a custom style has been applied to the column header **“Sold Amount”** under **“FY 2016”** via the [headerCellInfo](#) event and to the row header **“Germany”** and its aggregated value via the [queryCellInfo](#) event by adding the **“e-custom”** class to the cell element.

INDEX.JSX

```

import { PivotViewComponent } from '@syncfusion/ej2-react-pivotview';

```

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    dataSource: pivotData,
    expandAll: false,
    enableSorting: true,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  }
  let gridSettings = {
    queryCellInfo: function (args) {
      var colIndex = Number(args.cell.getAttribute('data-colindex'));
      var cells = args.data[colIndex] ? args.data[colIndex] : {};
      // Here by using 'actualText' option, a custom class can be added to
the specific row header and its value to apply custom style.
      if (cells.actualText === 'Germany') {
        args.cell.classList.add('e-custom');
      } else if (cells.actualText === 'Amount' &&
        cells.columnHeaders === 'FY 2016' && cells.rowHeaders === 'Germany')
{
        args.cell.classList.add('e-custom');
      }
    },
    headerCellInfo: function (args) {
      var customAttributes = args.cell.column.customAttributes;
      // Here custom class can be added to the specific column header by
using unique level name, to apply custom style.
      if (args.node.classList.contains('e-columnsheader') &&
customAttributes &&
        customAttributes.cell.valueSort.levelName === 'FY 2016.Sold Amount')
{
        args.node.classList.add('e-custom');
      }
    }
  };
  let pivotObj;
  return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings} width={'100%'} height={350}>
    </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';

```

```

import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: false,
    enableSorting: true,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  }
  let gridSettings: GridSettings = {
    queryCellInfo: function (args: any) {
      let colIndex: number = Number(args.cell.getAttribute('data-
colindex'));
      let cells: any = args.data[colIndex] ? args.data[colIndex] : {};
      // Here by using 'actualText' option, a custom class can be added to
the specific row header and its value to apply custom style.
      if (cells.actualText === 'Germany') {
        args.cell.classList.add('e-custom');
      } else if (cells.actualText === 'Amount' &&
cells.columnHeaders === 'FY 2016' && cells.rowHeaders === 'Germany')
{
        args.cell.classList.add('e-custom');
      }
    },
    headerCellInfo: function (args: any) {
      let customAttributes: any = args.cell.column.customAttributes;
      // Here custom class can be added to the specific column header by
using unique level name, to apply custom style.
      if (args.node.classList.contains('e-columnsheader') &&
customAttributes &&
customAttributes.cell.valueSort.levelName === 'FY 2016.Sold Amount')
{
        args.node.classList.add('e-custom');
      }
    }
  } as GridSettings;
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings} width={'100%'} height={350}>
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Note: The **dot(.)** character in **FY 2016.Sold Amount** is used by default to identify the header levels in the pivot table's row and column. It can be changed by setting the [headerDelimiter](#) in the [valueSortSettings](#) property to any other delimiter instead of the default separator.

Show tooltip for row and column headers in React Pivotview component

You can create and display the tooltip for each row and column header(s) in the pivot table by using an external tooltip component via the [dataBound](#) event.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
import { Tooltip } from '@syncfusion/ej2-popups';
function App() {
    let headerTooltip;
    let dataSourceSettings = {
        dataSource: pivotData,
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    let pivotObj;
    function dataBound() {
        if (!headerTooltip) {
            headerTooltip = new Tooltip({
                target: 'td.e-rowsheader,th.e-columnsheader', beforeRender: beforeRender
            });
            headerTooltip.appendTo(pivotObj.element);
        }
    }
    function beforeRender(args) {
        if (args.target.parentElement.querySelector('.e-rowsheader')) {
            // Here you can set custom content for row header(s) tooltip from its cell information.
            let index = Number(args.target.getAttributeNode('index').value);
            let colIndex = Number(args.target.getAttributeNode('data-colindex').value);
            let cell = pivotObj.engineModule.pivotValues[index][colIndex];
            let valueText = cell.valueSort ? cell.valueSort : '';
            if (cell.formattedText !== 'Grand Total') {
                this.content =
                    '<div>' +
                    'FieldName: ' +
                    valueText.axis +
                    '</br>' +
                    'Text: ' +
                    cell.formattedText +
            }
        }
    }
}
```

```

        '</div>';
    } else {
        this.content =
            '<div>' +
            'FieldName: ' +
            valueText.uniqueName +
            '</br>' +
            'Text: ' +
            cell.formattedText +
            '</div>';
    }
} else {
    // Here you can set custom content for column header(s) tooltip from
    // its cell information.
    if (args.target.querySelector('.e-cellvalue')) {
        this.content = args.target.querySelector('.e-cellvalue').innerText;
    } else if (args.target.querySelector('.e-headertext')) {
        this.content = args.target.querySelector('.e-headertext').innerText;
    } else if (args.target.querySelector('.e-stackedheadercelldiv')) {
        this.content = args.target.querySelector('.e-
stackedheadercelldiv').innerText;
    } else {
        this.content = '';
    }
}
}
}
return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'}
height={350} dataBound={dataBound.bind(this)}>
    <Inject services={[FieldList]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import {
    FieldList, IAxisSet, IDataOptions, IDataset, Inject, PivotViewComponent
} from '@syncfusion/ej2-react-pivotview';
import { Tooltip } from '@syncfusion/ej2-popups';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let headerTooltip: Tooltip;
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataset[],
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
    };
}

```

```

    filters: []
  }
  let pivotObj: PivotViewComponent;
  function dataBound(): void {
    if (!headerTooltip) {
      headerTooltip = new Tooltip({
        target: 'td.e-rowsheader,th.e-columnsheader', beforeRender:
beforeRender
      });
      headerTooltip.appendTo(pivotObj.element);
    }
  }
  function beforeRender(args: any) {
    if (args.target.parentElement.querySelector('.e-rowsheader')) {
      // Here you can set custom content for row header(s) tooltip from its
cell information.
      let index: number =
Number(args.target.getAttributeNode('index').value);
      let colIndex: number = Number(args.target.getAttributeNode('data-
colindex').value);
      let cell: IAxisSet =
pivotObj.engineModule.pivotValues[index][colIndex];
      let valueText: any = cell.valueSort ? cell.valueSort : '';
      if (cell.formattedText !== 'Grand Total') {
        this.content =
          '<div>' +
          'FieldName: ' +
          valueText.axis +
          '</br>' +
          'Text: ' +
          cell.formattedText +
          '</div>';
      } else {
        this.content =
          '<div>' +
          'FieldName: ' +
          valueText.uniqueName +
          '</br>' +
          'Text: ' +
          cell.formattedText +
          '</div>';
      }
    } else {
      // Here you can set custom content for column header(s) tooltip from
its cell information.
      if (args.target.querySelector('.e-cellvalue')) {
        this.content = args.target.querySelector('.e-cellvalue').innerText;
      } else if (args.target.querySelector('.e-headertext')) {
        this.content = args.target.querySelector('.e-headertext').innerText;
      } else if (args.target.querySelector('.e-stackedheadercelldiv')) {
        this.content = args.target.querySelector('.e-
stackedheadercelldiv').innerText;
      } else {
        this.content = '';
      }
    }
  }
}

```



```

    return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'}
    height={350} dataBound={dataBound.bind(this)}>
    <Inject services={[FieldList]} />
    </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Hide specific columns in React Pivotview component

By using the [columnRender](#) event in the [gridSettings](#), you can hide specific column(s) in the pivot table. In the example below, the “Units Sold” column under “FY 2016” is hidden by setting its **visible** property to **false** via the [columnRender](#) event.

Note: The **dot(.)** character in **FY 2016.Units Sold** is used by default to identify the header levels in the pivot table's row and column. It can be changed by setting the [headerDelimiter](#) in the [valueSortSettings](#) property to any other delimiter instead of the default separator.

INDEX.JSX

```

import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
    let dataSourceSettings = {
        dataSource: pivotData,
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    let gridSettings = {
        columnRender: function (args) {
            // Specific column(s) can be hidden by checking their level name and
            setting its visible property accordingly.
            for (let i = 1; i < args.columns.length; i++) {
                if (args.stackedColumns[i].customAttributes &&
                    args.stackedColumns[i].customAttributes.cell.valueSort.levelName
                    === 'FY 2016.Units Sold') {
                    args.stackedColumns[i].visible = false;
                }
            }
        }
    };
    let pivotObj;
    return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings}
    gridSettings={gridSettings} width={'100%'} height={350}>
    <Inject services={[FieldList]} />

```

```

    </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import {
  FieldList, IDataOptions, IDataset, Inject, PivotViewComponent
} from '@syncfusion/ej2-react-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: false,
    enableSorting: true,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  };
  let gridSettings: GridSettings = {
    columnRender: function (args: any) {
      // Specific column(s) can be hidden by checking their level name and
      setting its visible property accordingly.
      for (let i = 1; i < args.columns.length; i++) {
        if (args.stackedColumns[i].customAttributes &&
args.stackedColumns[i].customAttributes.cell.valueSort.levelName
=== 'FY 2016.Units Sold') {
          args.stackedColumns[i].visible = false;
        }
      }
    }
  }
  as GridSettings;
  let pivotObj: PivotViewComponent;
  return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings}
gridSettings={gridSettings} width={'100%'} height={350}>
    <Inject services={[FieldList]} />
  </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

<!-- markdownlint-disable MD009 -->

Export table and chart into the same document using toolbar in React Pivotview component. Even if the [displayOption.view](#) property is set to **Both** in the pivot table, you can only export either the table or the chart to the PDF document based on the current value set in the [displayOption.primary](#) property. But, to export both the table and the chart to the same PDF document, use the [pdfExport](#) method during the [actionBegin](#) event invoke.

In the following example, the built-in export action can be restricted by setting the [args.cancel](#) option to **true** in the [actionBegin](#) event, and both the table and the chart can be exported by calling the [pdfExport](#) method and setting the [exportBothTableAndChart](#) argument to **true**.

INDEX.JSX

```
import { FieldList, Inject, PivotViewComponent, Toolbar, PDFExport } from
 '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
 'Quarter' }],
    dataSource: pivotData,
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
 caption: 'Sold Amount' }]
  }
  let pivotObj;
  let toolbarOptions = ['Grid', 'Chart', 'Export', 'FieldList'];
  function actionBegin(args) {
    if (args.actionName === 'PDF export') {
      args.cancel = true;
      pivotObj.pdfExport({}, false, undefined, false, true);
    }
  }

  return (<PivotViewComponent ref={d => pivotObj = d } id='PivotView'
 height={350} showToolbar={true}
   dataSourceSettings={dataSourceSettings} showFieldList={true}
 allowPdfExport={true} enableVirtualization={true}
   displayOption={{ view: 'Both' }} toolbar={toolbarOptions}
 actionBegin={actionBegin.bind(this)}>
    <Inject services={[FieldList, Toolbar, PDFExport]} />
  </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { FieldList, IDataOptions, Inject, PivotViewComponent, IDataset,
 PivotActionBeginEventArgs, Toolbar, PDFExport } from '@syncfusion/ej2-react-
pivotview';
import * as React from 'react';
```

```

import * as ReactDOM from 'react-dom';
import { pivotData } from './datasource';
function App() {
  let dataSourceSettings: IDataOptions = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    dataSource: pivotData as IDataSet[],
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }]
  }
  let pivotObj: PivotViewComponent;
  let toolbarOptions: any = ['Grid', 'Chart', 'Export', 'FieldList'];
  function actionBegin(args: PivotActionBeginEventArgs): void {
    if (args.actionName === 'PDF export') {
      args.cancel = true;
      pivotObj.pdfExport({}, false, undefined, false, true);
    }
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} showToolbar={true}
  dataSourceSettings={dataSourceSettings} showFieldList={true}
allowPdfExport={true} enableVirtualization={true}
  displayOption={{ view: 'Both' }} toolbar={toolbarOptions}
actionBegin={actionBegin.bind(this)}>
    <Inject services={[FieldList, Toolbar, PDFExport]} />
  </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

<!-- markdownlint-disable MD009 -->

Add custom aggregation type to the menu in React Pivotview component

By using the [dataBound](#) event, you can add your own custom aggregate type(s) to the pivot table's aggregate menu.

In the following example, we have added the aggregation types **CustomAggregateType 1** and **CustomAggregateType 2** to the aggregate menu. The calculation for those aggregated types can be done using the [aggregateCellInfo](#) event.

INDEX.JSX

```

import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-
react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { L10n } from '@syncfusion/ej2-base';
import { pivotData } from './datasource';
L10n.load({
  'en-US': {
    pivotview: {
      CustomAggregateType1: 'Custom Aggregate Type 1',

```

```

        CustomAggregateType2: 'Custom Aggregate Type 2',
      },
      pivotfieldlist: {
        CustomAggregateType1: 'Custom Aggregate Type 1',
        CustomAggregateType2: 'Custom Aggregate Type 2',
      }
    }
  });
  const SummaryType = [
    'Sum',
    'Count',
    'DistinctCount',
    'Avg',
    'CustomAggregateType1',
    'CustomAggregateType2'
  ];
  function App() {
    let dataSourceSettings = {
      expandAll: false,
      dataSource: pivotData,
      columns: [{ name: 'Year' }, { name: 'Quarter' }],
      values: [{ name: 'Sold' }, { name: 'Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      subTotalsPosition: 'Bottom'
    };
    let pivotObj;
    function dataBound() {
      pivotObj.getAllSummaryType = function () {
        return SummaryType;
      };
      pivotObj.pivotFieldListModule.aggregateTypes = SummaryType;
      pivotObj.pivotFieldListModule.getAllSummaryType = function () {
        return SummaryType;
      };
    }
    function aggregateCell(args) {
      if (args.aggregateType === 'CustomAggregateType1') {
        args.value = args.value * 100;
      }
      if (args.aggregateType === 'CustomAggregateType2') {
        args.value = args.value / 100;
      }
    }
    return (
      <PivotViewComponent ref={d => pivotObj = d} id='PivotView'
        height={350} dataSourceSettings={dataSourceSettings}
        dataBound={dataBound.bind(this)} showFieldList={true}
        aggregateCellInfo={aggregateCell.bind(this)}>
        <Inject services={[FieldList]} />
      </PivotViewComponent>
    );
  }
  ;
  export default App;
  ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent,
AggregateEventArgs, AggregateTypes, SummaryTypes } from '@syncfusion/ej2-
react-pivotview';
import { L10n } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
L10n.load({
  'en-US': {
    pivotview: {
      CustomAggregateType1: 'Custom Aggregate Type 1',
      CustomAggregateType2: 'Custom Aggregate Type 2',
    },
    pivotfieldlist: {
      CustomAggregateType1: 'Custom Aggregate Type 1',
      CustomAggregateType2: 'Custom Aggregate Type 2',
    }
  }
});
const SummaryType: string[] = [
  'Sum',
  'Count',
  'DistinctCount',
  'Avg',
  'CustomAggregateType1',
  'CustomAggregateType2'
];
function App() {
  let dataSourceSettings: IDataOptions = {
    dataSource: pivotData as IDataset[],
    expandAll: false,
    columns: [{ name: 'Year' }, { name: 'Quarter' }],
    values: [{ name: 'Sold' }, { name: 'Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    subTotalsPosition: 'Bottom'
  }
  let pivotObj: PivotViewComponent;
  function dataBound(): void {
    pivotObj.getAllSummaryType = function () {
      return SummaryType as AggregateTypes[];
    };
    pivotObj.pivotFieldListModule.aggregateTypes = SummaryType as
AggregateTypes[];
    pivotObj.pivotFieldListModule.getAllSummaryType = function () {
      return SummaryType as AggregateTypes[];
    };
  }
  function aggregateCell(args: AggregateEventArgs): void {
    if (args.aggregateType === 'CustomAggregateType1' as SummaryTypes) {
      args.value = args.value as number * 100;
    }
    if (args.aggregateType === 'CustomAggregateType2' as SummaryTypes) {
      args.value = args.value as number / 100;
    }
  }
}

```

```

    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}
    dataBound={dataBound.bind(this)}
aggregateCellInfo={aggregateCell.bind(this)}>
    <Inject services={[FieldList]} />
    </PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

<!-- markdownlint-disable MD009 -->

Convert complex JSON to flat JSON and assign it to the pivot table in React Pivotview component. By default, flat JSON can only bind to the pivot table. However, you can connect complex JSON to the pivot table by converting it to flat JSON via code-behind and binding it to the pivot table using the [dataSource](#) property in the [load](#) event.

In the following example, the **complexToFlatJson()** method is used to convert complex JSON to flat JSON and bind it to the pivot table using the [dataSource](#) property, then modifying the field names in the [rows](#) and [columns](#) based on the converted flat JSON under [dataSourceSettings](#) in the [load](#) event.

INDEX.JSX

```

import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let dataSourceSettings = {
        expandAll: true,
        enableSorting: true,
        dataSource: data(),
        columns: [{ name: 'OrderDetails' }],
        values: [{ name: 'Freight', caption: 'Units Sold' }],
        rows: [{ name: 'ShipDetails' }],
        valueSortSettings: { headerDelimiter: ' - ' },
        formatSettings: [{ name: 'Amount', format: 'C0' }]
    }
    let pivotObj;
    let parentProp = {};
    let dataSource;
    function onLoad(args) {
        dataSource =
JSON.parse(JSON.stringify(args.dataSourceSettings.dataSource));
        args.dataSourceSettings.dataSource = complexToFlatJson(dataSource);
        let rows = [];
        for (let i = 0; i < args?.dataSourceSettings?.rows?.length; i++) {
            if (args.dataSourceSettings.rows[i].name in parentProp) {
                rows =
rows.concat(parentProp[args.dataSourceSettings.rows[i].name]);
            } else {
                rows.push(args.dataSourceSettings.rows[i]);
            }
        }
        args.dataSourceSettings.rows = rows;
        let columns = [];
    }
}

```

```

    for (let i = 0; i < args.dataSourceSettings.columns.length; i++) {
        if (args.dataSourceSettings.columns[i].name in parentProp) {
            columns = columns.concat(
                parentProp[args.dataSourceSettings.columns[i].name]
            );
        } else {
            columns.push(args.dataSourceSettings.columns[i]);
        }
    }
    args.dataSourceSettings.columns = columns;
}

function complexToFlatJson(data) {
    let flatArray = [];
    let flatObject = {};
    for (let index = 0; index < data.length; index++) {
        for (let prop in data[index]) {
            let value = data[index][prop];
            if (Array.isArray(value)) {
                for (let i = 0; i < value.length; i++) {
                    let childProp = [];
                    for (let inProp in value[i]) {
                        flatObject[inProp] = value[i][inProp];
                        let object = {
                            name: inProp,
                        };
                        childProp.push(object);
                    }
                    parentProp[prop] = childProp;
                }
            } else {
                flatObject[prop] = value;
            }
        }
        flatArray.push(flatObject);
        flatObject = {};
    }
    return flatArray;
}

function data() {
    return [
        {
            CustomerID: 'VINET',
            Freight: 32.38,
            OrderDetails: [
                {
                    OrderID: 10248,
                    OrderDate: '1996-07-04T10:10:00.000Z',
                }
            ],
            ShipDetails: [
                {
                    ShipName: 'Vins et alcools Chevalier',
                    ShipAddress: '59 rue de l'Abbaye',
                    ShipCity: 'Reims',
                    ShipRegion: null,
                    ShipCountry: 'France',
                    ShippedDate: '1996-07-16T12:20:00.000Z',
                }
            ]
        }
    ]
}

```



```

    }
  ]
},
{
  CustomerID: 'GALED',
  Freight: 10.14,
  OrderDetails: [
    {
      OrderID: 10366,
      OrderDate: '1996-11-28T00:00:00.000Z',
    }
  ],
  ShipDetails: [
    {
      ShippedDate: '1996-12-30T00:00:00.000Z',
      ShipName: 'Galería del gastronómo',
      ShipAddress: 'Rambla de Cataluña, 23',
      ShipCity: 'Barcelona',
      ShipRegion: null,
      ShipCountry: 'Spain',
    }
  ]
},
{
  CustomerID: 'VAFFE',
  Freight: 13.55,
  OrderDetails: [
    {
      OrderID: 10367,
      OrderDate: '1996-12-02T00:00:00.000Z',
    }
  ],
  ShipDetails: [
    {
      ShippedDate: '1996-12-30T00:00:00.000Z',
      ShipName: 'Vaffeljernet',
      ShipAddress: 'Smagsloget 45',
      ShipCity: 'Århus',
      ShipRegion: null,
      ShipCountry: 'Denmark',
    }
  ]
},
{
  CustomerID: 'ERNSH',
  Freight: 101.95,
  OrderDetails: [
    {
      OrderID: 10368,
      OrderDate: '1996-11-29T00:00:00.000Z',
    }
  ],
  ShipDetails: [
    {
      ShippedDate: '1996-12-30T00:00:00.000Z',
      ShipName: 'Ernst Handel',
      ShipAddress: 'Kirchgasse 6',

```

```

        ShipCity: 'Graz',
        ShipRegion: null,
        ShipCountry: 'Austria',
      }
    ],
  },
  {
    CustomerID: 'SPLIR',
    Freight: 195.68,
    OrderDetails: [
      {
        OrderID: 10369,
        OrderDate: '1996-11-28T00:00:00.000Z',
      }
    ],
    ShipDetails: [
      {
        ShippedDate: '1996-12-30T00:00:00.000Z',
        ShipName: 'Split Rail Beer & Ale',
        ShipAddress: 'P.O. Box 555',
        ShipCity: 'Lander',
        ShipRegion: 'WY',
        ShipCountry: 'USA',
      }
    ],
  }
];

return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350}
dataSourceSettings={dataSourceSettings} showFieldList={true}
load={onLoad.bind(this)}>
  <Inject services={[FieldList]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { FieldList, IDataOptions, Inject, PivotViewComponent } from
 '@syncfusion/ej2-react-pivotview';
import { enableRipple } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(false);
function App() {
  let dataSourceSettings: IDataOptions = {
    expandAll: true,
    enableSorting: true,
    dataSource: data() as any,
    columns: [{ name: 'OrderDetails' }],
    values: [{ name: 'Freight', caption: 'Units Sold' }],
    rows: [{ name: 'ShipDetails' }],
    valueSortSettings: { headerDelimiter: ' - ' },
    formatSettings: [{ name: 'Amount', format: 'C0' }]
  };

```

```

    }
    let pivotObj: PivotViewComponent;
    let parentProp: any = {};
    let dataSource: Object[][];
    function onLoad(args: any): void {
        dataSource =
JSON.parse(JSON.stringify(args.dataSourceSettings.dataSource));
        args.dataSourceSettings.dataSource = complexToFlatJson(dataSource);
        let rows: any = [];
        for (let i: number = 0; i < args.dataSourceSettings.rows.length; i++) {
            if (args.dataSourceSettings.rows[i].name in parentProp) {
                rows =
rows.concat(parentProp[args.dataSourceSettings.rows[i].name]);
            } else {
                rows.push(args.dataSourceSettings.rows[i]);
            }
        }
        args.dataSourceSettings.rows = rows;
        let columns: any = [];
        for (let i: number = 0; i < args.dataSourceSettings.columns.length; i++)
        {
            if (args.dataSourceSettings.columns[i].name in parentProp) {
                columns = columns.concat(
                    parentProp[args.dataSourceSettings.columns[i].name]
                );
            } else {
                columns.push(args.dataSourceSettings.columns[i]);
            }
        }
        args.dataSourceSettings.columns = columns;
    }
    function complexToFlatJson(data: Object[][]): {
        let flatArray: any = [];
        let flatObject: any = {};
        for (let index = 0; index < data.length; index++) {
            for (let prop in data[index]) {
                let value: Object = data[index][prop];
                if (Array.isArray(value)) {
                    for (let i: number = 0; i < value.length; i++) {
                        let childProp: any = [];
                        for (let inProp in value[i]) {
                            flatObject[inProp] = value[i][inProp];
                            let object = {
                                name: inProp,
                            };
                            childProp.push(object);
                        }
                        parentProp[prop] = childProp;
                    }
                } else {
                    flatObject[prop] = value;
                }
            }
            flatArray.push(flatObject);
            flatObject = {};
        }
        return flatArray;
    }

```

```

}
function data() {
  return [
    {
      CustomerID: 'VINET',
      Freight: 32.38,
      OrderDetails: [
        {
          OrderID: 10248,
          OrderDate: '1996-07-04T10:10:00.000Z',
        }
      ],
      ShipDetails: [
        {
          ShipName: 'Vins et alcools Chevalier',
          ShipAddress: '59 rue de l'Abbaye',
          ShipCity: 'Reims',
          ShipRegion: null,
          ShipCountry: 'France',
          ShippedDate: '1996-07-16T12:20:00.000Z',
        }
      ]
    },
    {
      CustomerID: 'GALED',
      Freight: 10.14,
      OrderDetails: [
        {
          OrderID: 10366,
          OrderDate: '1996-11-28T00:00:00.000Z',
        }
      ],
      ShipDetails: [
        {
          ShippedDate: '1996-12-30T00:00:00.000Z',
          ShipName: 'Galería del gastronómo',
          ShipAddress: 'Rambla de Cataluña, 23',
          ShipCity: 'Barcelona',
          ShipRegion: null,
          ShipCountry: 'Spain',
        }
      ]
    },
    {
      CustomerID: 'VAFFE',
      Freight: 13.55,
      OrderDetails: [
        {
          OrderID: 10367,
          OrderDate: '1996-12-02T00:00:00.000Z',
        }
      ],
      ShipDetails: [
        {
          ShippedDate: '1996-12-30T00:00:00.000Z',
          ShipName: 'Vaffeljernet',
          ShipAddress: 'Smagsloget 45',
        }
      ]
    }
  ]
}

```

```

        ShipCity: 'Århus',
        ShipRegion: null,
        ShipCountry: 'Denmark',
    }
]
},
{
    CustomerID: 'ERNSH',
    Freight: 101.95,
    OrderDetails: [
        {
            OrderID: 10368,
            OrderDate: '1996-11-29T00:00:00.000Z',
        }
    ],
    ShipDetails: [
        {
            ShippedDate: '1996-12-30T00:00:00.000Z',
            ShipName: 'Ernst Handel',
            ShipAddress: 'Kirchgasse 6',
            ShipCity: 'Graz',
            ShipRegion: null,
            ShipCountry: 'Austria',
        }
    ]
},
{
    CustomerID: 'SPLIR',
    Freight: 195.68,
    OrderDetails: [
        {
            OrderID: 10369,
            OrderDate: '1996-11-28T00:00:00.000Z',
        }
    ],
    ShipDetails: [
        {
            ShippedDate: '1996-12-30T00:00:00.000Z',
            ShipName: 'Split Rail Beer & Ale',
            ShipAddress: 'P.O. Box 555',
            ShipCity: 'Lander',
            ShipRegion: 'WY',
            ShipCountry: 'USA',
        }
    ],
}
];
}
}
return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350}
    dataSourceSettings={dataSourceSettings} showFieldList={true}
load={onLoad.bind(this)}>
    <Inject services={[FieldList]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

```
<!-- markdownlint-disable MD009 -->
```

Load desired report from the report list as default in React Pivotview component

By default, the pivot table is displayed with the report bound at the code-behind. To load a desired report from the previously saved report collection during initial rendering, set the desired report name in the [dataBound](#) event, along with the additional report-based customization code shown below.

INDEX.JSX

```
import {
  FieldList, Inject, PivotViewComponent, CalculatedField, Toolbar,
  ConditionalFormatting, NumberFormatting, PDFExport, ExcelExport
} from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import './App.css';
import { getInstance, select } from '@syncfusion/ej2-base';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { pivotData } from './datasource';
function App() {
  let isInitial = true;
  let dataSourceSettings = {
    dataSource: pivotData,
    columns: [{ name: 'Year' }, { name: 'Quarter' }],
    values: [{ name: 'Sold' }, { name: 'Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
  }
  let pivotObj;
  let toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove', 'Load',
    'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
    'ConditionalFormatting', 'NumberFormatting', 'FieldList'];
  function saveReport(args) {
    let reports = [];
    let isSaved = false;
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
      "") {
      reports = JSON.parse(localStorage.pivotviewReports);
    }
    if (args.report && args.reportName && args.reportName !== '' &&
      args.reportName !== 'Sample Report') {
      reports.map(function (item) {
        if (args.reportName === item.reportName) {
          item.report = args.report; isSaved = true;
        }
      });
      if (!isSaved) {
        reports.push(args);
      }
      localStorage.pivotviewReports = JSON.stringify(reports);
    }
  }
  function fetchReport(args) {
    let reportCollection = [];
    let reeportList = [];
```

```

    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item) {
reeportList.push(item.reportName); });
    args.reportName = reeportList;
}
function loadReport(args) {
    let reportCollection = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item) {
        if (args.reportName === item.reportName) {
            args.report = item.report;
        }
    });
    if (args.report) {
        pivotObj.dataSourceSettings =
JSON.parse(args.report).dataSourceSettings;
    }
}
function removeReport(args) {
    let reportCollection= [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    for (let i = 0; i < reportCollection.length; i++) {
        if (reportCollection[i].reportName === args.reportName) {
            reportCollection.splice(i, 1);
        }
    }
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        localStorage.pivotviewReports = JSON.stringify(reportCollection);
    }
}
function renameReport(args) {
    let reportCollection= [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    if (args.isReportExists) {
        for (let i = 0; i < reportCollection.length; i++) {
            if (reportCollection[i].reportName === args.rename) {
                reportCollection.splice(i, 1);
            }
        }
    }
    reportCollection.map(function (item) {
        if (args.reportName === item.reportName) {
            item.reportName = args.rename;
        }
    })
}

```

```

    });
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        localStorage.pivotviewReports = JSON.stringify(reportCollection);
    }
}
function dataBound() {
    if (pivotObj && isInitial) {
        isInitial = false;
        pivotObj.toolbarModule.action = 'Load';
        /* replace the report name by yours */
        let reportList = getInstance(select('#' + pivotObj.element.id +
        '_reportlist', pivotObj.element), DropDownList);
        reportList.value = 'Default report';
        loadReport({ reportName: 'Default report' });
    }
}
function load(){
    // Save the desired report that needs to be loaded at initial rendering
    here.
    let dataSourceSettings = {
        dataSource: pivotData,
        columns: [{ name: 'Year' }],
        enableSorting: true,
        allowLabelFilter: true,
        values: [{ name: 'Sold', caption: 'Units Sold' }],
        allowValueFilter: true,
        formatSettings: [{ name: 'Sold', format: 'C0' }],
        rows: [{ name: 'Country' }],
    };
    let displayOption = { view: 'Both' };
    let gridSettings = {columnWidth: 100};
    let report = { dataSourceSettings: dataSourceSettings, displayOption:
displayOption, gridSettings: gridSettings };
    let reports = [
        {
            report: JSON.stringify(report),
            reportName: 'Default report',
        },
    ];
    localStorage['pivotviewReports'] = JSON.stringify(reports);
}
function newReport() {
    pivotObj.setProperties({ dataSource: { columns: [], rows: [], values:
[], filters: [] } }, false);
}
function beforeToolbarRender(args) {
    args.customToolbar.splice(6, 0, {
        type: 'Separator'
    });
    args.customToolbar.splice(9, 0, {
        type: 'Separator'
    });
}
return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'}

```



```

        height={350} showFieldList={true} gridSettings={{ columnWidth: 140 }}
allowExcelExport={true}
        allowConditionalFormatting={true} allowNumberFormatting={true}
allowPdfExport={true} showToolbar={true}
        allowCalculatedField={true} displayOption={{ view: 'Both' }}
toolbar={toolbarOptions} newReport={newReport.bind(this)}
        renameReport={renameReport.bind(this)}
removeReport={removeReport.bind(this)} loadReport={loadReport.bind(this)}
        fetchReport={fetchReport.bind(this)} saveReport={saveReport.bind(this)}
toolbarRender={beforeToolbarRender.bind(this)}
dataBound={dataBound.bind(this)}
        load={load.bind(this)}>
    <Inject services={[FieldList, CalculatedField, Toolbar, PDFExport,
ExcelExport, ConditionalFormatting, NumberFormatting]} />
    </PivotViewComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import {
    FieldList, IDataOptions, Inject, PivotViewComponent, CalculatedField,
    Toolbar, RemoveReportArgs, ToolbarArgs,
    ConditionalFormatting, IDataSet, RenameReportArgs, SaveReportArgs,
    FetchReportArgs,
    LoadReportArgs, NumberFormatting, PDFExport, ExcelExport
} from '@syncfusion/ej2-react-pivotview';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { getInstance, select } from '@syncfusion/ej2-base';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { pivotData } from './datasource';
function App() {
    let isInitial: boolean = true;
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataSet[],
        columns: [{ name: 'Year' }, { name: 'Quarter' }],
        values: [{ name: 'Sold' }, { name: 'Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
    }
    let pivotObj: PivotViewComponent;
    let toolbarOptions: any = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'];
    function saveReport(args: any): void {
        let reports: SaveReportArgs[] = [];
        let isSaved: boolean = false;
        if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
'') {
            reports = JSON.parse(localStorage.pivotviewReports);
        }
        if (args.report && args.reportName && args.reportName !== '' &&
args.reportName !== 'Sample Report') {

```

```

        reports.map(function (item: any): any {
            if (args.reportName === item.reportName) {
                item.report = args.report; isSaved = true;
            }
        });
        if (!isSaved) {
            reports.push(args);
        }
        localStorage.pivotviewReports = JSON.stringify(reports);
    }
}

function fetchReport(args: FetchReportArgs): void {
    let reportCollection: string[] = [];
    let reeportList: string[] = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item: any): void {
        reeportList.push(item.reportName); });
    args.reportName = reeportList;
}

function loadReport(args: LoadReportArgs): void {
    let reportCollection: string[] = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    reportCollection.map(function (item: any): void {
        if (args.reportName === item.reportName) {
            args.report = item.report;
        }
    });
    if (args.report) {
        pivotObj.dataSourceSettings =
        JSON.parse(args.report).dataSourceSettings;
    }
}

function removeReport(args: RemoveReportArgs): void {
    let reportCollection: any[] = [];
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    for (let i: number = 0; i < reportCollection.length; i++) {
        if (reportCollection[i].reportName === args.reportName) {
            reportCollection.splice(i, 1);
        }
    }
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        localStorage.pivotviewReports = JSON.stringify(reportCollection);
    }
}

function renameReport(args: RenameReportArgs): void {
    let reportCollection: any[] = [];

```

```

    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        reportCollection = JSON.parse(localStorage.pivotviewReports);
    }
    if (args.isReportExists) {
        for (let i: number = 0; i < reportCollection.length; i++) {
            if (reportCollection[i].reportName === args.rename) {
                reportCollection.splice(i, 1);
            }
        }
    }
    reportCollection.map(function (item: any): any {
        if (args.reportName === item.reportName) {
            item.reportName = args.rename;
        }
    });
    if (localStorage.pivotviewReports && localStorage.pivotviewReports !==
    "") {
        localStorage.pivotviewReports = JSON.stringify(reportCollection);
    }
}
function dataBound(): void {
    if (pivotObj && isInitial) {
        isInitial = false;
        pivotObj.toolbarModule.action = 'Load';
        /* replace the report name by yours */
        let reportList = getInstance(select('#' + pivotObj.element.id +
        '_reportlist', pivotObj.element), DropDownList);
        (reportList as DropDownList).value = 'Default report';
        loadReport({ reportName: 'Default report' });
    }
}
function load(): void{
    // Save the desired report that needs to be loaded at initial rendering
    here.
    let dataSourceSettings = {
        dataSource: pivotData as IDataset[],
        columns: [{ name: 'Year' }],
        enableSorting: true,
        allowLabelFilter: true,
        values: [{ name: 'Sold', caption: 'Units Sold' }],
        allowValueFilter: true,
        formatSettings: [{ name: 'Sold', format: 'C0' }],
        rows: [{ name: 'Country' }],
    };
    let displayOption = { view: 'Both' };
    let gridSettings = {columnWidth: 100};
    let report = { dataSourceSettings: dataSourceSettings, displayOption:
    displayOption, gridSettings: gridSettings };
    let reports = [
        {
            report: JSON.stringify(report),
            reportName: 'Default report',
        },
    ];
    localStorage['pivotviewReports'] = JSON.stringify(reports);
}

```

```

function newReport(): void {
  pivotObj.setProperties({ dataSource: { columns: [], rows: [], values:
  [], filters: [] } }, false);
}
function beforeToolbarRender(args: any): void {
  args.customToolbar.splice(6, 0, {
    type: 'Separator'
  });
  args.customToolbar.splice(9, 0, {
    type: 'Separator'
  });
}
return (<PivotViewComponent id='PivotView' ref={d => pivotObj = d}
dataSourceSettings={dataSourceSettings} width={'100%'}
height={350} showFieldList={true} gridSettings={{ columnWidth: 140 }}
allowExcelExport={true}
allowConditionalFormatting={true} allowNumberFormatting={true}
allowPdfExport={true} showToolbar={true}
allowCalculatedField={true} displayOption={{ view: 'Both' }}
toolbar={toolbarOptions} newReport={newReport.bind(this)}
renameReport={renameReport.bind(this)}
removeReport={removeReport.bind(this)} loadReport={loadReport.bind(this)}
fetchReport={fetchReport.bind(this)} saveReport={saveReport.bind(this)}
toolbarRender={beforeToolbarRender.bind(this)}
dataBound={dataBound.bind(this)}
load={load.bind(this)}>
  <Inject services={[FieldList, CalculatedField, Toolbar, PDFExport,
  ExcelExport, ConditionalFormatting, NumberFormatting]} />
</PivotViewComponent>;
);
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

<!-- markdownlint-disable MD009 -->

Display string value to pivot table values in React Pivotview component

End user can display string value to the pivot table's value cell by using the [aggregateCellInfo](#) event.

In the following example, each cell value of the **Sold** field's actual value has been assigned from its combination data sets obtained from the [args.cellSets](#) in the [aggregateCellInfo](#) event.

INDEX.JSX

```

import { FieldList, Inject, PivotViewComponent } from '@syncfusion/ej2-react-pivotview';
import { enableRipple } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
enableRipple(false);
function App() {
  let dataSourceSettings = {
    expandAll: false,
    dataSource: pivotData,
    columns: [{ name: 'Year' }, { name: 'Quarter' }],
    values: [{ name: 'Sold' }, { name: 'Amount' }],

```

```

        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        subTotalsPosition: 'Bottom'
    };
    let pivotObj;
    function aggregateCell(args) {
        if (args.fieldName === 'Sold') {
            args.value = secondsToHms(args.value);
        }
    }
    function secondsToHms(d) {
        d = Number(d);
        var h = Math.floor(d / 3600);
        var m = Math.floor((d % 3600) / 60);
        var s = Math.floor((d % 3600) % 60);
        return (
            ('0' + h).slice(-2) + ':' + ('0' + m).slice(-2) + ':' + ('0' +
s).slice(-2)
        );
    }
    return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}
    aggregateCellInfo={aggregateCell.bind(this)}>
        <Inject services={[FieldList]} />
    </PivotViewComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { FieldList, IDataOptions, IDataset, Inject, PivotViewComponent }
from '@syncfusion/ej2-react-pivotview';
import { enableRipple } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { pivotData } from '../datasource';
enableRipple(false);
function App() {
    let dataSourceSettings: IDataOptions = {
        dataSource: pivotData as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year' }, { name: 'Quarter' }],
        values: [{ name: 'Sold' }, { name: 'Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        subTotalsPosition: 'Bottom'
    }
    let pivotObj: PivotViewComponent;
    function aggregateCell(args: any): void {
        if (args.fieldName === 'Sold') {
            args.value = secondsToHms(args.value);
        }
    }
    function secondsToHms(d: number) {

```

```

    d = Number(d);
    let h = Math.floor(d / 3600);
    let m = Math.floor((d % 3600) / 60);
    let s = Math.floor((d % 3600) % 60);
    return (
      ('0' + h).slice(-2) + ':' + ('0' + m).slice(-2) + ':' + ('0' +
s).slice(-2)
    );
  }
  return (<PivotViewComponent ref={d => pivotObj = d} id='PivotView'
height={350} dataSourceSettings={dataSourceSettings} showFieldList={true}
aggregateCellInfo={aggregateCell.bind(this)}>
  <Inject services={[FieldList]} />
</PivotViewComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

Summary of Predefined Dialogs component

Predefined dialogs

Getting Started

This section explain how to create the predefined dialogs in React application with its basic features in step-by-step procedure.

Dependencies

The following list of dependencies are required to use the React Dialog component in your application.

```
`javascript
```

```
|-- @syncfusion/ej2-react-popups
```

```
|-- @syncfusion/ej2-react-base
```

```
|-- @syncfusion/ej2-react-buttons
```

```
|-- @syncfusion/ej2-popups
```

```
|-- @syncfusion/ej2-base
```

```
|-- @syncfusion/ej2-buttons
```

```
,
```

Setup your development environment

You can use [Create-react-app](#) to setup the applications.

To install `create-react-app` run the following command.

```
`bash
```

```
npm install -g create-react-app
```

```
,
```

Start a new project using create-react-app command as follows

```
<div class='tsx'>
`

create-react-app quickstart --scripts-version=react-scripts-ts
cd quickstart
`

</div>

<div class='jsx'>
`

create-react-app quickstart
cd quickstart
`

</div>
```

'react-scripts-ts' is used for creating React app with typescript.

Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) public registry.

You can choose the component that you want to install. For this application, we are going to use Dialog component.

To install Dialog component, use the following command

```
`bash
npm install @syncfusion/ej2-react-popups --save
`
```

Adding CSS reference

Import the Dialog component's required CSS references as follows in `src/App.css`.

```
`css
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-react-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-react-popups/styles/material.css";
`
```

The [Custom Resource Generator \(CRG\)](#) is an online web tool, which can be used to generate the custom script and styles for a set of specific components.

This web tool is useful to combine the required component scripts and styles in a single file.

Render a dialog using utility functions

The dialog component provides built-in utility functions to render the alert and confirm dialogs with the minimal code.

The following options are used as an argument on calling the utility functions:

Options	Description
title	Specifies the title of dialog like the header property.
content	Specifies the value that can be displayed in dialog's content area like the content property.
isModal	Specifies the Boolean value whether the dialog can be displayed as modal or non-modal. For more details, refer to the isModal property.
position	Specifies the value where the alert or confirm dialog is positioned within the document. For more details, refer to the position property { X: 'center', Y: 'center' }
okButton	Configures the OK button that contains button properties with the click events. okButton:{ icon:'prefix icon to the button', cssClass:'custom class to the button', click: 'action for OK button click', text: 'Yes' // <-- Default value is 'OK' }
cancelButton	Configures the Cancel button that contains button properties with the click events. cancelButton:{ icon:'prefix icon to the button', cssClass:'custom class to the button', click: 'action for 'Cancel' button click', text: 'No' // <-- Default value is 'Cancel' }
isDraggable	Specifies the value whether the alert or confirm dialog can be dragged by the user.
showCloseIcon	When set to true, the close icon is shown in the dialog component.
closeOnEscape	When set to true, you can close the dialog by pressing ESC key.
animationSettings	Specifies the animation settings of the dialog component.
cssClass	Specifies the CSS class name that can be appended to the dialog.
zIndex	Specifies the order of the dialog, that is displayed in front or behind of another component.
open	Event which is triggered after the dialog is opened.
Close	Event which is triggered after the dialog is closed.

Adding predefined dialogs to the application

Now, you can start adding React predefined dialog to the application. We have added predefined dialog component in `src/App.tsx` file using following code.

```
[Class-component]
```

```
`ts
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {hideDialog: boolean;}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
}
```



```
public buttonClick() {
  dialogObj = DialogUtility.alert({
    title: 'Low Battery',
    width: '250px',
    content: '10% of battery remaining'
  });
}

public render() {
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
        onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
}

export default App;
`ts

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }

  buttonClick() {
    dialogObj = DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining'
    });
  }
}
```

```

render() {
return (<div className="App" id='dialog-target'>
<ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
</div>);
}
}
export default App;
`

```

[Functional-component]

```

`ts
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App(){
function buttonClick() {
dialogObj = DialogUtility.alert({
title: 'Low Battery',
width: '250px',
content: '10% of battery remaining'
});
}
return (
<div className="App" id='dialog-target'>
<ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
</div>
);
}
export default App;
`
`ts
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';

```

```
import * as React from "react";

function App() {
  function buttonClick() {
    dialogObj = DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining'
    });
  }

  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="alertBtn" cssClass="e-danger"
      onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
  </div>);
}

export default App;
```

Run the application

Now use the `npm run start` command to run the application in the browser.

```
npm run start
```

The below example shows the alert dialog.

[Class-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    dialogObj = DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining'
    });
  }
  render() {
```

```

        return (<div className="App" id='dialog-target'>
            <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
        </div>);
    }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
    constructor(props: {}) {
        super(props);
        this.state = { };
    }
    public buttonClick() {
        dialogObj = DialogUtility.alert({
            title: 'Low Battery',
            width: '250px',
            content: '10% of battery remaining'
        });
    }
    public render() {
        return (
            <div className="App" id='dialog-target'>
                <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
            </div>);
    }
}
export default App;

```

[Functional-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        dialogObj = DialogUtility.alert({
            title: 'Low Battery',
            width: '250px',
            content: '10% of battery remaining'
        });
    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
}
export default App;

```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        dialogObj = DialogUtility.alert({
            title: 'Low Battery',
            width: '250px',
            content: '10% of battery remaining'
        });
    }
    return (
        <div className="App" id='dialog-target'>
            <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
        </div>
    );
}
export default App;
```

Alert dialog

An alert dialog box used to display an errors, warnings, and information alerts that needs user awareness. The alert dialog is displayed along with the OK button. When user clicks on 'OK' button, alert dialog will get closed. Use the following code to render a simple alert dialog in an application.

[Class-component]**APP.JSX**

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
let dialogObj;
class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = {};
    }
    buttonClick() {
        document.getElementById("statusText").style.display = "none";
        dialogObj = DialogUtility.alert({
            title: 'Low Battery',
            width: '250px',
            content: '10% of battery remaining',
            okButton: { click: this.alertOkAction.bind(this) },
        });
    }
    alertOkAction() {
        dialogObj.hide();
        document.getElementById('statusText').innerHTML =
            'The user closed the Alert dialog.';
        document.getElementById('statusText').style.display = 'block';
    }
}
```

```

    }
    render() {
        return (<div className="App" id='dialog-target'>
            <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
            <span id="statusText"></span>
        </div>);
    }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
let dialogObj;
class App extends React.Component<{}, {}> {
    constructor(props: {}) {
        super(props);
        this.state = { };
    }
    public buttonClick() {
        document.getElementById("statusText").style.display = "none";
        dialogObj = DialogUtility.alert({
            title: 'Low Battery',
            width: '250px',
            content: '10% of battery remaining',
            okButton: { click: this.alertOkAction.bind(this) },
        });
    }
    public alertOkAction() {
        dialogObj.hide();
        document.getElementById('statusText').innerHTML =
            'The user closed the Alert dialog.';
        document.getElementById('statusText').style.display = 'block';
    }
    public render() {
        return (
            <div className="App" id='dialog-target'>
                <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
                <span id="statusText"></span>
            </div>);
    }
}
export default App;

```

[Functional-component]**APP.JSX**

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
let dialogObj;

```

```
function App() {
  function buttonClick() {
    document.getElementById("statusText").style.display = "none";
    dialogObj = DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      okButton: { click: alertOkAction.bind(this) },
    });
  }
  function alertOkAction() {
    dialogObj.hide();
    document.getElementById('statusText').innerHTML =
      'The user closed the Alert dialog.';
    document.getElementById('statusText').style.display = 'block';
  }
  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    <span id="statusText"></span>
  </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
let dialogObj;
function App() {
  function buttonClick() {
    document.getElementById("statusText").style.display = "none";
    dialogObj = DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      okButton: { click: alertOkAction.bind(this) },
    });
  }
  function alertOkAction() {
    dialogObj.hide();
    document.getElementById('statusText').innerHTML =
      'The user closed the Alert dialog.';
    document.getElementById('statusText').style.display = 'block';
  }
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
      <span id="statusText"></span>
    </div>
  );
}
export default App;
```

Confirm dialog

A confirm dialog box used to displays a specified message along with the 'OK' and 'Cancel' button. It is used to get approval from the user, and it appears before any critical action. After get approval from the user the dialog will disappear automatically. Use the following code to render a simple confirm dialog in an application.

[Class-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
let dialogObj;
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    document.getElementById("statusText").style.display = "none";
    dialogObj = DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      okButton: { click: this.confirmOkAction.bind(this) },
      cancelButton: { click: this.confirmCancelAction.bind(this) }
    });
  }
  confirmOkAction() {
    dialogObj.hide();
    document.getElementById("statusText").innerHTML = " The user
confirmed the dialog box";
    document.getElementById("statusText").style.display = "block";
  }
  confirmCancelAction() {
    dialogObj.hide();
    document.getElementById("statusText").innerHTML = "The user canceled
the dialog box.";
    document.getElementById("statusText").style.display = "block";
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
      <span id="statusText"></span>
    </div>);
  }
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
```



```

import * as React from "react";
let dialogObj;
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    document.getElementById("statusText").style.display = "none";
    dialogObj = DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these items?',
      width: '300px',
      okButton: { click: this.confirmOkAction.bind(this) },
      cancelButton: { click: this.confirmCancelAction.bind(this) }
    });
  }
  public confirmOkAction() {
    dialogObj.hide();
    document.getElementById("statusText").innerHTML = " The user confirmed the dialog box";
    document.getElementById("statusText").style.display = "block";
  }
  public confirmCancelAction() {
    dialogObj.hide();
    document.getElementById("statusText").innerHTML = "The user canceled the dialog box.";
    document.getElementById("statusText").style.display = "block";
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
        <span id="statusText"></span>
      </div>);
  }
}
export default App;

```

[Functional-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
let dialogObj;
function App() {
  function buttonClick() {
    document.getElementById("statusText").style.display = "none";
    dialogObj = DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',

```

```

        okButton: { click: confirmOkAction.bind(this) },
        cancelButton: { click: confirmCancelAction.bind(this) }
    });
}
function confirmOkAction() {
    dialogObj.hide();
    document.getElementById("statusText").innerHTML = " The user
confirmed the dialog box";
    document.getElementById("statusText").style.display = "block";
}
function confirmCancelAction() {
    dialogObj.hide();
    document.getElementById("statusText").innerHTML = "The user canceled
the dialog box.";
    document.getElementById("statusText").style.display = "block";
}
return (<div className="App" id='dialog-target'>
    <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
    <span id="statusText"></span>
</div>);
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
let dialogObj;
function App() {
    function buttonClick() {
        document.getElementById("statusText").style.display = "none";
        dialogObj = DialogUtility.confirm({
            title: 'Delete Multiple Items',
            content: 'Are you sure you want to permanently delete these
items?',
            width: '300px',
            okButton: { click: confirmOkAction.bind(this) },
            cancelButton: { click: confirmCancelAction.bind(this) }
        });
    }
    function confirmOkAction() {
        dialogObj.hide();
        document.getElementById("statusText").innerHTML = " The user
confirmed the dialog box";
        document.getElementById("statusText").style.display = "block";
    }
    function confirmCancelAction() {
        dialogObj.hide();
        document.getElementById("statusText").innerHTML = "The user
canceled the dialog box.";
        document.getElementById("statusText").style.display = "block";
    }
    return (
        <div className="App" id='dialog-target'>

```

```

        <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
        <span id="statusText"></span>
    </div>
    );
}
export default App;

```

Prompt dialog

A prompt dialog is used to get the input from the user. When the user clicks the 'OK' button the input value from the dialog is returned. If the user clicks the 'Cancel' button the null value is returned. After getting the input from the user the dialog will disappear automatically.

[Class-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
let dialogObj;
class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = {};
    }
    buttonClick() {
        document.getElementById("statusText").style.display = "none";
        dialogObj = DialogUtility.confirm({
            title: 'Join Chat Group',
            width: '300px',
            content: '<p>Enter your name:</p> <input id= "inputEle"
type="text" name="Required" class="e-input" placeholder="Type here.." />',
            okButton: { click: this.promptOkAction.bind(this) },
            cancelButton: { click: this.promptCancelAction.bind(this) },
        });
    }
    promptOkAction() {
        let value;
        value = document.getElementById("inputEle").value;
        if (value == "") {
            dialogObj.hide();
            document.getElementById("statusText").innerHTML = "The user's
input is returned as\" \" ";
            document.getElementById("statusText").style.display = "block";
        }
        else {
            dialogObj.hide();
            document.getElementById("statusText").innerHTML = "The user's
input is returned as" + " " + value;
            document.getElementById("statusText").style.display = "block";
        }
    }
    promptCancelAction() {
        dialogObj.hide();
    }
}

```

```

        document.getElementById("statusText").innerHTML = "The user canceled
the prompt dialog";
        document.getElementById("statusText").style.display = "block";
    }
    render() {
        return (<div className="App" id='dialog-target'>
            <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
            <span id="statusText"></span>
        </div>);
    }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
let dialogObj;
class App extends React.Component<{}, {}> {
    constructor(props: {}) {
        super(props);
        this.state = { };
    }
    public buttonClick() {
        document.getElementById("statusText").style.display = "none";
        dialogObj = DialogUtility.confirm({
            title: 'Join Chat Group',
            width: '300px',
            content:
                '<p>Enter your name:</p> <input id= "inputEle" type="text"
name="Required" class="e-input" placeholder="Type here.." />',
            okButton: { click: this.promptOkAction.bind(this) },
            cancelButton: { click: this.promptCancelAction.bind(this) },
        });
    }
    public promptOkAction() {
        let value:string ;
        value = (document.getElementById("inputEle") as any).value;
        if (value=="") {
            dialogObj.hide();
            document.getElementById("statusText").innerHTML = "The user's input
is returned as\" \" ";
            document.getElementById("statusText").style.display="block";
        }
        else{
            dialogObj.hide();
            document.getElementById("statusText").innerHTML="The user's input is
returned as" + " " + value;
            document.getElementById("statusText").style.display="block";
        }
    }
    public promptCancelAction() {
        dialogObj.hide();
    }
}

```

```

        document.getElementById("statusText").innerHTML="The user canceled
the prompt dialog";
        document.getElementById("statusText").style.display="block";
    }
    public render() {
        return (
            <div className="App" id='dialog-target'>
                <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
                <span id="statusText"></span>
            </div>);
    }
}
export default App;

```

[Functional-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
let dialogObj;
function App() {
    function buttonClick() {
        document.getElementById("statusText").style.display = "none";
        dialogObj = DialogUtility.confirm({
            title: 'Join Chat Group',
            width: '300px',
            content: '<p>Enter your name:</p> <input id= "inputEle"
type="text" name="Required" class="e-input" placeholder="Type here.." />',
            okButton: { click: promptOkAction.bind(this) },
            cancelButton: { click: promptCancelAction.bind(this) },
        });
    }
    function promptOkAction() {
        let value;
        value = document.getElementById("inputEle").value;
        if (value == "") {
            dialogObj.hide();
            document.getElementById("statusText").innerHTML = "The user's
input is returned as\" \" ";
            document.getElementById("statusText").style.display = "block";
        }
        else {
            dialogObj.hide();
            document.getElementById("statusText").innerHTML = "The user's
input is returned as" + " " + value;
            document.getElementById("statusText").style.display = "block";
        }
    }
    function promptCancelAction() {
        dialogObj.hide();
        document.getElementById("statusText").innerHTML = "The user canceled
the prompt dialog";
        document.getElementById("statusText").style.display = "block";
    }
}

```

```

    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
        <span id="statusText"></span>
    </div>);
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
let dialogObj;
function App() {
    function buttonClick() {
        document.getElementById("statusText").style.display = "none";
        dialogObj = DialogUtility.confirm({
            title: 'Join Chat Group',
            width: '300px',
            content:
                '<p>Enter your name:</p> <input id= "inputEle" type="text"
name="Required" class="e-input" placeholder="Type here.." />',
            okButton: { click: promptOkAction.bind(this) },
            cancelButton: { click: promptCancelAction.bind(this) },
        });
    }
    function promptOkAction() {
        let value:string ;
        value = (document.getElementById("inputEle") as any).value;
        if (value=="") {
            dialogObj.hide();
            document.getElementById("statusText").innerHTML = "The user's
input is returned as\" \" ";
            document.getElementById("statusText").style.display="block";
        }
        else{
            dialogObj.hide();
            document.getElementById("statusText").innerHTML="The user's
input is returned as" + " " + value;
            document.getElementById("statusText").style.display="block";
        }
    }
    function promptCancelAction() {
        dialogObj.hide();
        document.getElementById("statusText").innerHTML="The user
canceled the prompt dialog";
        document.getElementById("statusText").style.display="block";
    }
    return (
        <div className="App" id='dialog-target'>
            <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
            <span id="statusText"></span>
        </div>
    );
}

```

```
);
}
export default App;
```

Draggable in React Predefined dialogs component

The predefined dialogs supports dragging within its target container by grabbing the dialog header, which allows the user to reposition the dialog dynamically by using `isDraggable` property.

Alert dragging

[Class-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      isDraggable: true
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
  }
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      isDraggable : true
    });
  }
}
```

```

}
public render() {
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
  }
}
export default App;

```

[Functional-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      isDraggable: true
    });
  }
  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      isDraggable : true
    });
  }
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>
  );
}
export default App;

```


Confirm drag

[Class-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      isDraggable: true
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>);
  }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these items?',
      width: '300px',
      isDraggable : true
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
      </div>);
  }
}

```

```

    }
  }
  export default App;

```

[functional-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      isDraggable: true
    });
  }
  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>);
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      isDraggable : true
    });
  }
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>
  );
}
export default App;

```

Prompt drag

[Class-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      width: '300px',
      content: '<p>Enter your name:</p> <input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
      isDraggable: true
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
    </div>);
  }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      width: '300px',
      content: '<p>Enter your name:</p> <input type="text" name="Required"
class="e-input" placeholder="Type here.." />',
      isDraggable : true
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
      </div>);
  }
}
export default App;

```

[Functional-component]**APP.JSX**

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Join Chat Group',
            width: '300px',
            content: '<p>Enter your name:</p> <input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
            isDraggable: true
        });
    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
    </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Join Chat Group',
            width: '300px',
            content: '<p>Enter your name:</p> <input type="text" name="Required"
class="e-input" placeholder="Type here.." />',
            isDraggable : true
        });
    }
    return (
        <div className="App" id='dialog-target'>
            <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
        </div>
    );
}
export default App;
```

Animation in React Predefined dialogs component

The predefined dialogs can be animated during the open and close actions. Also, user can customize animation's **delay**, **duration** and **effect** of animation by using the **animationSettings** property.

In the below sample, **Zoom** effect is enabled. So, The Dialog will open with **ZoomIn** and close with **ZoomOut** effects.

Alert animation

[Class-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      animationSettings: { effect: 'Zoom' }
    });
  }
  render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
          onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
      </div>);
  }
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      animationSettings: { effect: 'Zoom' }
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
          onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
      </div>);
  }
}
```

```

    </div>);
  }
}
export default App;

```

[Functional-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      animationSettings: { effect: 'Zoom' }
    });
  }
  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      animationSettings: { effect: 'Zoom' }
    });
  }
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>
  );
}
export default App;

```

Confirm animation

[Class-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      animationSettings: { effect: 'Zoom' }
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>);
  }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these items?',
      width: '300px',
      animationSettings: { effect: 'Zoom' }
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
      </div>);
  }
}
export default App;

```

[Functionla-component]**APP.JSX**

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Delete Multiple Items',
            content: 'Are you sure you want to permanently delete these
items?',
            width: '300px',
            animationSettings: { effect: 'Zoom' }
        });
    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
        </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Delete Multiple Items',
            content: 'Are you sure you want to permanently delete these
items?',
            width: '300px',
            animationSettings: { effect: 'Zoom' }
        });
    }
    return (
        <div className="App" id='dialog-target'>
            <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
        </div>
    );
}
export default App;
```

Prompt animation

[Class-component]**APP.JSX**

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
```



```

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      width: '300px',
      content: '<p>Enter your name:</p> <input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
      animationSettings: { effect: 'Zoom' }
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
    </div>);
  }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      width: '300px',
      content:
        '<p>Enter your name:</p> <input type="text" name="Required"
class="e-input" placeholder="Type here.." />',
      animationSettings: { effect: 'Zoom' }
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
      </div>);
  }
}
export default App;

```

[Functional-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Join Chat Group',
            width: '300px',
            content: '<p>Enter your name:</p> <input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
            animationSettings: { effect: 'Zoom' }
        });
    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
        </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Join Chat Group',
            width: '300px',
            content:
                '<p>Enter your name:</p> <input type="text" name="Required"
class="e-input" placeholder="Type here.." />',
            animationSettings: { effect: 'Zoom' }
        });
    }
    return (
        <div className="App" id='dialog-target'>
            <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
            </div>
    );
}
export default App;
```

Position in React Predefined dialogs component

Customize the dialog position by using the **position** property. The position can be represented with specific **X** and **Y** values.

- The **PositionDataModel.X** can be configured with a left, center, right, or offset value. By default, the value is set as **center**.

- The `PositionDataModel.Y` can be configured with a top, center, bottom, or offset value. By default, the value is set as `center`.

Alert position

[Class-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      position: { X: 'center', Y: 'center' }
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
  }
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      position: { X: 'center', Y: 'center' }
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
```

```

    <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>;
  }
}
export default App;

```

[Functional-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      position: { X: 'center', Y: 'center' }
    });
  }
  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App(){
  function buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      position: { X: 'center', Y: 'center' }
    });
  }
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>
  );
}
export default App;

```

Confirm position

[Class-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      position: { X: 'center', Y: 'center' }
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>);
  }
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these items?',
      width: '300px',
      position: { X: 'center', Y: 'center' }
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
      </div>);
  }
}
```

```
}
export default App;
```

[Functional-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Delete Multiple Items',
            content: 'Are you sure you want to permanently delete these
items?',
            width: '300px',
            position: { X: 'center', Y: 'center' }
        });
    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
        </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Delete Multiple Items',
            content: 'Are you sure you want to permanently delete these
items?',
            width: '300px',
            position: { X: 'center', Y: 'center' }
        });
    }
    return (
        <div className="App" id='dialog-target'>
            <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
        </div>
    );
}
export default App;
```

Prompt position

[Class-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      width: '300px',
      content: '<p>Enter your name:</p><input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
      position: { X: 'center', Y: 'center' }
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
    </div>);
  }
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      width: '300px',
      content: '<p>Enter your name:</p><input type="text" name="Required"
class="e-input" placeholder="Type here.." />',
      position: { X: 'center', Y: 'center' }
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
      </div>);
  }
}
export default App;
```

[Functional-component]**APP.JSX**

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Join Chat Group',
            width: '300px',
            content: '<p>Enter your name:</p><input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
            position: { X: 'center', Y: 'center' }
        });
    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
        </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Join Chat Group',
            width: '300px',
            content: '<p>Enter your name:</p><input type="text" name="Required"
class="e-input" placeholder="Type here.." />',
            position: { X: 'center', Y: 'center' }
        });
    }
    return (
        <div className="App" id='dialog-target'>
            <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
        </div>
    );
}
export default App;
```

[Dimension in React Predefined dialogs component](#)

Customize the predefined dialogs dimensions using the **height** and **width** properties. You can specify the dimension values in both pixels and percentage format to change the default dialog width and height values.

[Alert dimension](#)**[Class-component]**

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      content: '10% of battery remaining',
      width: '250px',
      height: '200px'
    });
  }
  render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
          onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
      </div>);
  }
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      content: '10% of battery remaining',
      width : '250px',
      height: '200px'
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
          onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
      </div>);
  }
}
export default App;
```

[Functional-component]**APP.JSX**

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.alert({
            title: 'Low Battery',
            content: '10% of battery remaining',
            width: '250px',
            height: '200px'
        });
    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
        </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.alert({
            title: 'Low Battery',
            content: '10% of battery remaining',
            width : '250px',
            height: '200px'
        });
    }
    return (
        <div className="App" id='dialog-target'>
            <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
            </div>
        );
}
export default App;
```

Confirm dimension

[Class-component]**APP.JSX**

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
    constructor(props) {
```

```

    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      height: '200px'
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>);
  }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these items?',
      width: '300px',
      height: '200px'
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
      </div>);
  }
}
export default App;

```

[Functional-component]**APP.JSX**

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";

```

```
function App() {
  function buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      height: '200px'
    });
  }
  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      height: '200px'
    });
  }
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>
  );
}
export default App;
```

Prompt dimension

[Class-component]**APP.JSX**

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
```

```

        DialogUtility.confirm({
            title: 'Join Chat Group',
            content: '<p>Enter your name:</p> <input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
            height: '250px',
            width: '300px'
        });
    }
    render() {
        return (<div className="App" id='dialog-target'>
            <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
        </div>);
    }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
    constructor(props: {}) {
        super(props);
        this.state = { };
    }
    public buttonClick() {
        DialogUtility.confirm({
            title: 'Join Chat Group',
            content: '<p>Enter your name:</p> <input type="text" name="Required"
class="e-input" placeholder="Type here.." />',
            height: '250px',
            width: '300px'
        });
    }
    public render() {
        return (
            <div className="App" id='dialog-target'>
                <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
            </div>);
    }
}
export default App;

```

[Functional-component]**APP.JSX**

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({

```

```

        title: 'Join Chat Group',
        content: '<p>Enter your name:</p> <input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
        height: '250px',
        width: '300px'
    });
}
return (<div className="App" id='dialog-target'>
    <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
</div>);
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Join Chat Group',
            content: '<p>Enter your name:</p> <input type="text" name="Required"
class="e-input" placeholder="Type here.." />',
            height: '250px',
            width: '300px'
        });
    }
    return (
        <div className="App" id='dialog-target'>
            <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
        </div>
    );
}
export default App;

```

Max-width and max-height

To have a restricted max-width and max-height dialog dimension, you need to specify the max-width, max-height CSS properties for the component's container element by using the `cssClass` property. The max-height value is calculated in source level and set to the dialog. so, need to override the max-height property.

Use the following code to customize the max-width and max-height for alert dialog:

[Class-component]**APP.JSX**

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
    constructor(props) {

```

```

    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.alert({
      title: 'About SYNCFUSION Succinctly Serires',
      content: 'In the Succinctly series, Syncfusion created a robust,
free library of more than 130 technical e-books formatted for PDF, Kindle,
and EPUB.Each title in the Succinctly series is written by a carefully
chosen expert and provides essential content in about 100 pages. The
Succinctly series was born in 2012 out of a desire to provide concise
technical e-books for software developers.',
      cssClass: 'customClass'
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
  }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.alert({
      title: 'About SYNCFUSION Succinctly Serires',
      content: 'In the Succinctly series, Syncfusion created a robust,
free library of more than 130 technical e-books formatted for PDF, Kindle,
and EPUB.Each title in the Succinctly series is written by a carefully
chosen expert and provides essential content in about 100 pages. The
Succinctly series was born in 2012 out of a desire to provide concise
technical e-books for software developers.',
      cssClass : 'customClass'
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
      </div>);
  }
}
export default App;

```

[Functional-component]**APP.JSX**

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.alert({
            title: 'About SYNCFUSION Succinctly Serires',
            content: 'In the Succinctly series, Syncfusion created a robust,
free library of more than 130 technical e-books formatted for PDF, Kindle,
and EPUB.Each title in the Succinctly series is written by a carefully
chosen expert and provides essential content in about 100 pages. The
Succinctly series was born in 2012 out of a desire to provide concise
technical e-books for software developers.',
            cssClass: 'customClass'
        });
    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
        </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.alert({
            title: 'About SYNCFUSION Succinctly Serires',
            content: 'In the Succinctly series, Syncfusion created a robust,
free library of more than 130 technical e-books formatted for PDF, Kindle,
and EPUB.Each title in the Succinctly series is written by a carefully
chosen expert and provides essential content in about 100 pages. The
Succinctly series was born in 2012 out of a desire to provide concise
technical e-books for software developers.',
            cssClass : 'customClass'
        });
    }
    return (
        <div className="App" id='dialog-target'>
            <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
        </div>
    );
}
export default App;
```


Min-width and min-height

To have a restricted min-width and min-height dialog dimension, you need to specify the min-width, min-height CSS properties for the component's container element by using the `cssClass` property.

Use the following code to customize the min-width and min-height for alert dialog:

[Class-Component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.alert({
      title: 'About SYNCFUSION Succinctly Series',
      content: ' The Succinctly series was born in 2012 out of a
desire to provide concise technical e-books for software developers.',
      cssClass: 'customClass'
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
  }
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.alert({
      title: 'About SYNCFUSION Succinctly Series',
      content: ' The Succinctly series was born in 2012 out of a desire to
provide concise technical e-books for software developers.',
      cssClass : 'customClass'
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
```

```

    <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>;
  }
}
export default App;

```

[Functional-Component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.alert({
      title: 'About SYNCFUSION Succinctly Serires',
      content: ' The Succinctly series was born in 2012 out of a
desire to provide concise technical e-books for software developers.',
      cssClass: 'customClass'
    });
  }
  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.alert({
      title: 'About SYNCFUSION Succinctly Serires',
      content: ' The Succinctly series was born in 2012 out of a
desire to provide concise technical e-books for software developers.',
      cssClass : 'customClass'
    });
  }
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>
  );
}
export default App;

```

Customization in React Predefined dialogs component

You can customize the predefined dialogs buttons by using below properties.

- **okButton** - Use this property to customize **OK** button text.
- **cancelButton** - Use this property to customize **Cancel** button text.

Use the following code snippet for **alert**, **confirm** and **prompt** to customize the predefined dialogs action buttons.

For alert dialog , customized the default dialog button content as **Dismiss** by using the **text** property.

For confirm dialog, customized the default dialog buttons content as **Yes** and **No** by using the **text** property and also customized the dialog button icons by using **icon** property.

For prompt dialog , customized the default dialog buttons content as **Connect** and **Close** by using **text** property.

Alert action button

[Class-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      content: '10% of battery remaining',
      width: '250px',
      okButton: { text: 'Dismiss' }
    });
  }
  render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
          onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
      </div>);
  }
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
  }
}
```

```

        this.state = { };
    }
    public buttonClick() {
        DialogUtility.alert({
            title: 'Low Battery',
            content: '10% of battery remaining',
            width : '250px',
            okButton: { text: 'Dismiss' }
        });
    }
    public render() {
        return (
            <div className="App" id='dialog-target'>
                <ButtonComponent id="alertBtn" cssClass="e-danger"
                    onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
            </div>);
    }
}
export default App;

```

[Functional-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.alert({
            title: 'Low Battery',
            content: '10% of battery remaining',
            width: '250px',
            okButton: { text: 'Dismiss' }
        });
    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
            onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>);
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.alert({
            title: 'Low Battery',
            content: '10% of battery remaining',
            width : '250px',
            okButton: { text: 'Dismiss' }
        });
    }
}

```

```

    }
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
      </div>
    );
  }
}
export default App;

```

Confirm action buttons

[Class-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      okButton: { text: 'Yes', icon: 'e-icons e-check' },
      cancelButton: { text: 'No', icon: 'e-icons e-close' }
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>);
  }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',

```

```

        content: 'Are you sure you want to permanently delete these items?',
        width: '300px',
        okButton: { text: 'Yes', icon: 'e-icons e-check' },
        cancelButton: { text: 'No', icon: 'e-icons e-close' }
    });
}
public render() {
    return (
        <div className="App" id='dialog-target'>
            <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
        </div>);
    }
}
export default App;

```

[Functional-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Delete Multiple Items',
            content: 'Are you sure you want to permanently delete these
items?',
            width: '300px',
            okButton: { text: 'Yes', icon: 'e-icons e-check' },
            cancelButton: { text: 'No', icon: 'e-icons e-close' }
        });
    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>);
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Delete Multiple Items',
            content: 'Are you sure you want to permanently delete these
items?',
            width: '300px',
            okButton: { text: 'Yes', icon: 'e-icons e-check' },
            cancelButton: { text: 'No', icon: 'e-icons e-close' }
        });
    }
}

```

```

    }
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
      </div>
    );
  }
}
export default App;

```

Prompt action buttons

[Class-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content: '<p>Enter your name:</p><input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
      height: '200px',
      okButton: { text: 'Connect' },
      cancelButton: { text: 'Close' }
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
    </div>);
  }
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',

```

```

        content:
            '<p>Enter your name:</p><input type="text" name="Required" class="e-
input" placeholder="Type here.." />',
            height: '200px',
            okButton: { text: 'Connect' },
            cancelButton: { text: 'Close' }
        });
    }
    public render() {
        return (
            <div className="App" id='dialog-target'>
                <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
            </div>);
    }
}
export default App;

```

[Functional-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Join Chat Group',
            content: '<p>Enter your name:</p><input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
            height: '200px',
            okButton: { text: 'Connect' },
            cancelButton: { text: 'Close' }
        });
    }
    return (<div className="App" id='dialog-target'>
        <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
    </div>);
}
export default App;

```

APP.TSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
    function buttonClick() {
        DialogUtility.confirm({
            title: 'Join Chat Group',
            content:
                '<p>Enter your name:</p><input type="text" name="Required"
class="e-input" placeholder="Type here.." />',
            height: '200px',

```



```

        okButton: { text: 'Connect' },
        cancelButton: { text: 'Close' }
    });
}
return (
    <div className="App" id='dialog-target'>
        <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
    </div>
);
}
export default App;

```

Show or hide dialog close button

When rendering the predefined dialogs through utility methods, You can close the dialog using the following ways. The default values of `closeOnEscape` and `showCloseIcon` is `false`.

- By pressing the escape key if the `closeOnEscape` property is enabled.
- By clicking the close button if the `showCloseIcon` property is enabled.

You can also manually close the Dialogs by creating an instance to the dialog and call the `hide` method.

Use the following code for **alert**, **confirm** and **prompt** to demonstrates the different ways of hiding the utility dialog.

Alert dialog close button

[Class-component]

APP.JSX

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = {};
    }
    buttonClick() {
        DialogUtility.alert({
            title: 'Low Battery',
            width: '250px',
            content: '10% of battery remaining',
            showCloseIcon: true,
            closeOnEscape: true
        });
    }
    render() {
        return (<div className="App" id='dialog-target'>
            <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
        </div>);
    }
}
export default App;

```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      showCloseIcon : true,
      closeOnEscape : true
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={this.buttonClick.bind(this)}>Alert</ButtonComponent>
      </div>);
  }
}
export default App;
```

[Functional-component]**APP.JSX**

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      showCloseIcon: true,
      closeOnEscape: true
    });
  }
  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
  </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.alert({
      title: 'Low Battery',
      width: '250px',
      content: '10% of battery remaining',
      showCloseIcon : true,
      closeOnEscape : true
    });
  }
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="alertBtn" cssClass="e-danger"
onClick={buttonClick.bind(this)}>Alert</ButtonComponent>
    </div>
  );
}
export default App;
```

Confirm dialog close button

[Class-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      showCloseIcon: true,
      closeOnEscape: true
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>);
  }
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these items?',
      width: '300px',
      showCloseIcon : true,
      closeOnEscape : true
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={this.buttonClick.bind(this)}>Confirm</ButtonComponent>
      </div>);
  }
}
export default App;
```

[Functional-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      showCloseIcon: true,
      closeOnEscape: true
    });
  }
  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
  </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
```

```
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these
items?',
      width: '300px',
      showCloseIcon : true,
      closeOnEscape : true
    });
  }
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="confirmBtn" cssClass="e-success"
onClick={buttonClick.bind(this)}>Confirm</ButtonComponent>
    </div>
  );
}
export default App;
```

Prompt dialog close button

[Class-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content: '<p>Enter your name:</p> <input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
      width: '300px',
      showCloseIcon: true,
      closeOnEscape: true
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
    </div>);
  }
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component<{}>, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content:
        '<p>Enter your name:</p> <input type="text" name="Required"
class="e-input" placeholder="Type here.." />',
      width: '300px',
      showCloseIcon : true,
      closeOnEscape : true
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
      </div>);
  }
}
export default App;
```

[Functional-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content: '<p>Enter your name:</p> <input type="text"
name="Required" class="e-input" placeholder="Type here.." />',
      width: '300px',
      showCloseIcon: true,
      closeOnEscape: true
    });
  }
  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
  </div>);
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content:
        '<p>Enter your name:</p> <input type="text" name="Required"
class="e-input" placeholder="Type here.." />',
      width: '300px',
      showCloseIcon : true,
      closeOnEscape : true
    });
  }
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
    </div>
  );
}
export default App;
```

Customize dialog content

You can load custom content in predefined dialogs using the `content` property.

Use the following code to customize the dialog content to render the custom TextBox component inside the prompt dialog to get the username from the user.

[Class-component]

APP.JSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }
  buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content: '<p>Enter your name:</p><input class="e-input"
placeholder="Type here.." />',
      width: '300px',
    });
  }
  render() {
    return (<div className="App" id='dialog-target'>
      <ButtonComponent id="promptBtn" isPrimary
onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
    </div>);
  }
}
```

```
}
export default App;
```

APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import { TextBoxComponent } from '@syncfusion/ej2-react-inputs';
import * as React from "react";
class App extends React.Component<{}, {}> {
  constructor(props: {}) {
    super(props);
    this.state = { };
  }
  public buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content:
        '<p>Enter your name:</p><input class="e-input" placeholder="Type here.." />',
      width: '300px',
    });
  }
  public render() {
    return (
      <div className="App" id='dialog-target'>
        <ButtonComponent id="promptBtn" isPrimary
          onClick={this.buttonClick.bind(this)}>Prompt</ButtonComponent>
      </div>);
  }
}
export default App;
```

[Functional-component]**APP.JSX**

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content: '<p>Enter your name:</p><input class="e-input"
placeholder="Type here.." />',
      width: '300px',
    });
  }
  return (<div className="App" id='dialog-target'>
    <ButtonComponent id="promptBtn" isPrimary
      onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
  </div>);
}
export default App;
```


APP.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogUtility } from '@syncfusion/ej2-react-popups';
import { TextBoxComponent } from '@syncfusion/ej2-react-inputs';
import * as React from "react";
function App() {
  function buttonClick() {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content:
        '<p>Enter your name:</p><input class="e-input" placeholder="Type here.." />',
      width: '300px',
    });
  }
  return (
    <div className="App" id='dialog-target'>
      <ButtonComponent id="promptBtn" isPrimary
onClick={buttonClick.bind(this)}>Prompt</ButtonComponent>
    </div>
  );
}
export default App;
```

Progress bar

Getting Started

This section explains the steps required to create the ProgressBar control using React and configure its properties.

Dependencies

Below is the list of minimum dependencies required to use the progress bar component.

```
`javascript
|-- @syncfusion/ej2-react-progressbar
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-svg-base
`,`
```

Installation and configuration

You can use [create-react-app](#) to setup the applications.

To install `create-react-app` run the following command.

```
`
npm install -g create-react-app
`,`
```

- To setup basic `React` sample use following commands.

`

```
create-react-app quickstart --template typescript
```

```
cd quickstart
```

```
npm start
```

`

- Install Syncfusion packages using below command.

`

```
npm install @syncfusion/ej2-react-progressbar --save
```

`

Add Progressbar to the Project

Now, you can start adding Progress bar component in the application. For getting started, add the Progress bar component in `src/App.tsx` file using following code.

```
`ts
```

```
import {ProgressBarComponent} from '@syncfusion/ej2-react-progressbar';
```

```
import * as React from 'react';
```

```
function App() {
```

```
  return ( <ProgressBarComponent/>)
```

```
};
```

```
export default App;
```

```
`
```

Run the application

Now, we might create a simple progress bar sample as shown below.

INDEX.JSX

```
{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  return (
    <ProgressBarComponent id="linear" type='Linear' height='60'
    value={40} animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }}>
      </ProgressBarComponent>
    );
}
;
```

```
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  return (<ProgressBarComponent id="linear"
    type='Linear'
    height='60'
    value={40}
    animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }}>
    </ProgressBarComponent>
  );
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

Now run the `npm start` command in the console, it will run your application and open the browser window.

,

npm start

,

Types in React Progress bar component

Visualize progress in different shapes (rectangle, circle, and semi-circle) to give a unique appearance to your app design.

Linear

Set **type** to Linear to get the linear progress bar. It also support secondary progress and different mode of progress.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
function App() {
  return (<ProgressBarComponent id="lineardeterminate" type="Linear"
    height="60" value={100} animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }}>
    </ProgressBarComponent>
  );
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

```

    }}>
    </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
    return (<ProgressBarComponent
        id="lineardeterminate"
        type="Linear"
        height="60"
        value={100}
        animation={{
            enable: true,
            duration: 2000,
            delay: 0
        }}>
        </ProgressBarComponent>
    );
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

Circular

Set **type** to Circular to get the circular progress bar. It also support secondary progress and different mode of progress.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
    return (<ProgressBarComponent id="circular-container" type="Circular"
        height="160px" value={100} animation={{
            enable: true,
            duration: 2000,
            delay: 0
        }}>
        </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
  return (<ProgressBarComponent
    id="circular-container"
    type="Circular"
    height="160px"
    value={100}
    animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }}>
    </ProgressBarComponent>
  );
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

Tooltip in React Progress bar component

Tooltip

The tooltip for the progress bar is used to represent the progress value. During the initial load, it can be enabled by using the [enable](#) property. The [showTooltipOnHover](#) property can show the tooltip on mouseover.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
  return (<ProgressBarComponent id="lineardeterminate" type="Linear"
    height="60" tooltip={{enable: true} value={100} animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }}>
    </ProgressBarComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
```

```
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
    return (<ProgressBarComponent
        id="lineardeterminate"
        type="Linear"
        height="60"
        value={100}
        animation={{
            enable: true,
            duration: 2000,
            delay: 0
        }}
        tooltip={{enable: true}}>
    </ProgressBarComponent>
    );
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

Format

By default, the tooltip shows information about progress. In addition to that, show more information in the tooltip using the [format](#) property.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
    return (<ProgressBarComponent id="lineardeterminate" type="Linear"
height="60" tooltip={{enable: true, format: "Progress: ${value}%"}}
value={100} animation={{
    enable: true,
    duration: 2000,
    delay: 0
}}>
    </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
    return (<ProgressBarComponent
```

```

        id="lineardeterminate"
        type="Linear"
        height="60"
        value={100}
        animation={{
          enable: true,
          duration: 2000,
          delay: 0
        }}
        tooltip={{enable: true, format: "Progress: ${value}"}}>
      </ProgressBarComponent>
    )
  };
  export default App;
  ReactDOM.render(<App />, document.getElementById("container"));
  {% endraw %}

```

Customization

The [fill](#) and [border](#) properties are used to customize the background color and border of the tooltip respectively. The [textStyle](#) property in the tooltip is used to customize the font of the tooltip text.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
  return (<ProgressBarComponent id="lineardeterminate" type="Linear"
    height="60" value={100} animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }} tooltip={{
      enable: true,
      textStyle: {
        fontWeight: '600',
        size: '9px',
        color: 'red',
        fontFamily: 'Roboto',
        fontStyle: 'Italic'
      }
    }}>
    </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";

```

```
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
  return (<ProgressBarComponent
    id="lineardeterminate"
    type="Linear"
    height="60"
    value={100}
    animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }}
    tooltip={{enable: true,
      textStyle: {
        fontWeight: '600',
        size: '9px',
        color: 'red',
        fontFamily: 'Roboto',
        fontStyle: 'Italic'
      }}}
  </ProgressBarComponent>
  )
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

States in React Progress bar component

Visualize progress in different modes.

Determinate

<!-- markdownlint-disable MD033 -->

This is the default state. You can use it when the progress estimation is known.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
  return (<ProgressBarComponent id="linear" type="Linear" height="60"
value={100} animation={{
  enable: true,
  duration: 2000,
  delay: 0
}}>
  </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```


INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
    return (<ProgressBarComponent
        id="linear"
        type="Linear"
        height="60"
        value={100}
        animation={{
            enable: true,
            duration: 2000,
            delay: 0
        }}>
    </ProgressBarComponent>
    );
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

Indeterminate

By enabling the **isIndeterminate** property, the state of the progress bar can be changed to indeterminate when the progress cannot be estimated or is not being calculated. It can be combined with determinate mode to know that the application is estimating progress before the actual progress starts.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
    return (<ProgressBarComponent id="linear" type="Linear" height="60"
    value={20} isIndeterminate={true} animation={{
        enable: true,
        duration: 2000,
        delay: 0
    }}>
    </ProgressBarComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
```

```
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
  return (<ProgressBarComponent
    id="linear"
    type="Linear"
    height="60"
    value={20}
    isIndeterminate={true}
    animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }}>
    </ProgressBarComponent>
  );
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

Buffer

<!-- markdownlint-disable MD033 -->

You can use a secondary progress indicator when the primary progress depends on the secondary progress. This will allow users to visualize both primary and secondary progress simultaneously.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
  return (<ProgressBarComponent id="linear" type="Linear" height="60"
    value={40} secondaryProgress={60} animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }}>
    </ProgressBarComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ProgressBarComponent } from "@syncfusion/ej2-react-progressbar";
function App() {
  return (<ProgressBarComponent
    id="linear"
    type="Linear"

```

```

        height="60"
        value={40}
        secondaryProgress={60}
        animation={{
          enable: true,
          duration: 2000,
          delay: 0
        }}
      </ProgressBarComponent>
    )
  };
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

Customization in React Progress bar component

Segments

We can divide a progress bar into multiple segments using a `segmentCount` to visualize the progress of multiple sequential tasks.

INDEX.JSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  return (<ProgressBarComponent id="circular" type='Circular'
    height='160px' segmentCount={8} value={100} animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }}
  ></ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  return(<ProgressBarComponent id="circular"
    type='Circular'
    height='160px'
    segmentCount={8}
    value={100}
    animation={{
      enable: true,
      duration: 2000,

```

```

                delay: 0
            }>
        </ProgressBarComponent>
    )
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

Thickness

Customize the thickness of the track using [trackThickness](#), progress using [progressThickness](#) and secondary progress using [secondaryProgressThickness](#) to render the progress bar with different appearances.

INDEX.JSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
    return (<ProgressBarComponent id="circular" type='Circular'
height='160px' width='90%' trackThickness={24} progressThickness={24}
secondaryProgressThickness={20} value={100} animation={{
        enable: true,
        duration: 2000,
        delay: 0
    }}>
    </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
    return (<ProgressBarComponent id="circular"
        type='Circular'
        height='160px'
        width='90%'
        trackThickness={24}
        progressThickness={24}
        secondaryProgressThickness={20}
        value={100}
        animation={{
            enable: true,
            duration: 2000,
            delay: 0
        }}>
    </ProgressBarComponent>);
}

```

```

    </ProgressBarComponent>
  )
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

Radius

The radius of the progress bar can be customized using **radius** property and corner can be customized by **cornerRadius** property.

INDEX.JSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  return (<ProgressBarComponent id="circular" type='Circular'
    height='160px' width='90%' trackThickness={80} progressThickness={10}
    value={100} enableRtl={false} trackColor="#FFD939" radius="100%"
    progressColor="white" cornerRadius="Round" animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }}>
    </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  return (<ProgressBarComponent id="circular"
    type='Circular'
    height='160px'
    width='90%'
    trackThickness={80}
    progressThickness={10}
    value={100}
    enableRtl={false}
    trackColor="#FFD939"
    radius="100%"
    progressColor="white"
    cornerRadius="Round"
    animation={{
      enable: true,
      duration: 2000,

```

```

                delay: 0
            }>
        </ProgressBarComponent>
    )
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

InnerRadius

The inner radius of the progress bar can be customized using `innerRadius` property.

INDEX.JSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
    return (<ProgressBarComponent id="circular" type='Circular'
        height='160px' width='90%' trackThickness={80} progressThickness={10}
        value={100} enableRtl={false} trackColor="#FFD939" radius="100%"
        innerRadius='80%' progressColor="white" cornerRadius="Round" animation={{
            enable: true,
            duration: 2000,
            delay: 0
        }}>
        </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
    return (<ProgressBarComponent id="circular"
        type='Circular'
        height='160px'
        width='90%'
        trackThickness={80}
        progressThickness={10}
        value={100}
        enableRtl={false}
        trackColor="#FFD939"
        radius="100%"
        innerRadius='80%'
        progressColor="white"
        cornerRadius="Round"
        animation={{

```

```

        enable: true,
        duration: 2000,
        delay: 0
      }}>
    </ProgressBarComponent>
  )
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

Progress color and track color

Customize the color of progress, secondary progress, and track by using the [progressColor](#), [secondaryProgressColor](#), and [trackColor](#) properties.

INDEX.JSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  return (<ProgressBarComponent id="circular" type='Circular'
    height='160px' width='90%' trackThickness={24} progressThickness={24}
    value={50} secondaryProgress={70} secondaryProgressColor='green'
    enableRtl={false} showProgressValue={true} trackColor="#F8C7D8"
    radius="100%" progressColor="#E3165B" cornerRadius="Round" animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }}>
    </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  return (
    <ProgressBarComponent id="circular"
      type='Circular'
      height='160px'
      width='90%'
      trackThickness={24}
      progressThickness={24}
      secondaryProgressColor='green'
      value={50}
      enableRtl={false}

```

```

        showProgressValue={true}
        trackColor="#F8C7D8"
        radius="100%"
        progressColor="#E3165B"
        cornerRadius="Round"
        secondaryProgress={70}
        animation={{
            enable: true,
            duration: 2000,
            delay: 0
        }}
    </ProgressBarComponent>
)
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

Annotation in React Progress bar component

Annotation

In the circular progress bar, you can add any view to the center using the **Content** property in annotation.

For example, you can include add, start, or pause button to control the progress. You can also add an image that indicates the actual task in progress or add custom text that conveys how far the task is completed.

INDEX.JSX

```

import { ProgressBarComponent, ProgressBarAnnotationsDirective,
ProgressBarAnnotationDirective, Inject, ProgressAnnotation } from
'@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
    let content = '<div id="point1" style="font-size:20px;font-
weight:bold;color:#ffffff;fill:#ffffff"><span>60%</span></div>';
    return (<ProgressBarComponent id="circular" type='Circular'
innerRadius="190%" height='160px' trackThickness={80} cornerRadius={"Round"}
trackColor={"#FFD939"}>
        <Inject services={[ProgressAnnotation]}>/>
        <ProgressBarAnnotationsDirective>
            <ProgressBarAnnotationDirective content={content}>
                </ProgressBarAnnotationDirective>
            </ProgressBarAnnotationsDirective>
        </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));

```

INDEX.TSX

```

import { ProgressBarComponent, ProgressBarAnnotationsDirective,
ProgressBarAnnotationDirective, Inject,

```



```

ProgressAnnotation } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  let content: string = '<div id="point1" style="font-size:20px;font-weight:bold;color:#ffffff;fill:#ffffff"><span>60%</span></div>';
  return (<ProgressBarComponent id="circular" type='Circular'
innerRadius="190%" height='160px' trackThickness={80} cornerRadius={"Round"}
trackColor={"#FFD939"}>
    <Inject services={[ProgressAnnotation]} />
    <ProgressBarAnnotationsDirective>
      <ProgressBarAnnotationDirective content={content}>
        </ProgressBarAnnotationDirective>
      </ProgressBarAnnotationsDirective>
    </ProgressBarComponent>
  )
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));

```

Label

You can show the progress value in both linear and circular progress bar using **showProgressValue** property.

INDEX.JSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  return (<ProgressBarComponent id="linear" type='Linear'
showProgressValue={true} labelStyle={{ color: '#FFFFFF' }}
trackThickness={24} progressThickness={24} value={50} textRender={ (args) =>
{
  args.text = '50';
}} animation={{
  enable: true,
  duration: 2000,
  delay: 0,
}}>
    </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {

```

```

    return( <ProgressBarComponent id="linear"
        type='Linear'
        showProgressValue={true}
        labelStyle={{color: '#FFFFFF'}}
        trackThickness={24}
        progressThickness={24}
        value={50}
        textRender={(args: ITextRenderEventArgs) => {
            args.text = '50';
        }}
        animation={{
            enable: true,
            duration: 2000,
            delay: 0,
        }}>
    </ProgressBarComponent>
    )
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

Animation in React Progress bar component

Progress Bar support to animate the progress by using **animation** property. Enable the animation by setting **enable** property and also you can control the speed by using **duration** property.

INDEX.JSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
    return (<ProgressBarComponent id="linear" type='Linear'
        trackThickness={24} progressThickness={24} value={60} animation={{
            enable: true,
            duration: 2000,
            delay: 0
        }}>
    </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
    return (
        <ProgressBarComponent id="linear"

```

```

        type='Linear'
        trackThickness={24}
        progressThickness={24}
        value={60}
        animation={{
            enable: true,
            duration: 2000,
            delay: 0
        }}
    </ProgressBarComponent>
)
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

Range in React Progress bar component

Range represents the entire span of the progress bar and can be defined using the **minimum** and **maximum** properties. The default value of the range is 0 to 100.

INDEX.JSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
    return (<ProgressBarComponent id="Circular" type='Circular'
        height='160px' trackThickness={24} progressThickness={24} minimum={0}
        maximum={1} value={0.5} animation={{
            enable: true,
            duration: 2000,
            delay: 0
        }}
    ></ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
    return (<ProgressBarComponent id="Circular"
        type='Circular'
        height='160px'
        trackThickness={24}
        progressThickness={24}
        minimum={0}
        maximum={1}
    ></ProgressBarComponent>);
}

```

```

        value={0.5}
        animation={{
          enable: true,
          duration: 2000,
          delay: 0
        }}
      </ProgressBarComponent>
    )
  };
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

Events in React Progress bar component

valueChanged

This event is triggered when the progress value is changed.

INDEX.JSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  let progressInstance;
  function changeValue() {
    progressInstance.value = 80;
  }
  return (<div> <button onClick={changeValue}>Change
value</button><br></div>
    <ProgressBarComponent id="linear" ref={linear1 =>
progressInstance = linear1 type='Linear' trackColor='gray'
progressColor='blue' value={100} animation={{
  enable: false,
  duration: 2000,
  delay: 0
}} valueChanged={ (args) => {
  args.progressValue = 90;
}}>
    </ProgressBarComponent>
    </div>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import { IProgressValueEventArgs } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {

```

```

let progressInstance: ProgressBarComponent;
function changeValue() {
  progressInstance.value = 80;
}
return(<div> <button onClick={changeValue}>Change value</button><br></br>
  <ProgressBarComponent id="linear"
    ref={linear1 => progressInstance = linear1}
    type='Linear'
    trackColor='gray'
    progressColor='blue'
    value={100}
    animation={{
      enable: false,
      duration: 2000,
      delay: 0
    }}
    valueChanged={({ args: IProgressValueEventArgs) => {
      args.progressValue = 90;
    }}>
  </ProgressBarComponent>
</div>
)
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% enddraw %}

```

progressCompleted

This event is triggered when the progress attains the maximum value.

INDEX.JSX

```

{% raw %}
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  return (<ProgressBarComponent id="linear2" type='Linear' value={100}
    animation={{
      enable: true,
      duration: 2000,
      delay: 0
    }} progressCompleted={({ args) => {
      args.value = 50;
    }}>
  </ProgressBarComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% enddraw %}

```

INDEX.TSX

```
{% raw %}
```

```
import { ProgressBarComponent } from '@syncfusion/ej2-react-progressbar';
import { IProgressValueEventArgs } from '@syncfusion/ej2-progressbar';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
    return(<ProgressBarComponent id="linear2"
        type='Linear'
        value={100}
        animation={{
            enable: true,
            duration: 2000,
            delay: 0
        }}
        progressCompleted={(args: IProgressValueEventArgs)
=> {
            args.value = 50;
        }}>
        </ProgressBarComponent>
    );
};
export default App;
ReactDOM.render(<App />, document.getElementById("container"));
{% endraw %}
```

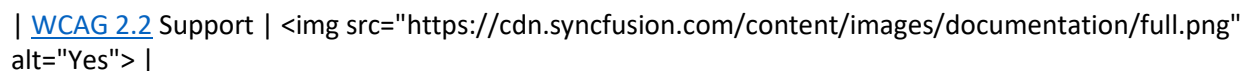
Accessibility in React Progress bar component

The Progress bar component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Progress bar component is outlined below.

| Accessibility Criteria | Compatibility |

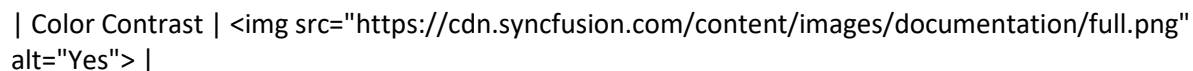
| -- | -- |

| [WCAG 2.2](#) Support |  alt="Yes" |

| [Section 508](#) Support |  alt="Yes" |

| Screen Reader Support |  alt="Yes" |

| Right-To-Left Support |  alt="Yes" |

| Color Contrast |  alt="Yes" |

| Mobile Device Support |  alt="Yes" |

| Keyboard Navigation Support |  alt="Yes" |

| [Accessibility Checker](#) Validation |  alt="Yes" |

```
| Axe-core Accessibility Validation |  |
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>
<div> - All
features of the component meet the requirement.</div>
<div> - Some features of the component do not meet the requirement.</div>
<div> - The component does not meet the requirement.</div>
```

WAI-ARIA attributes

The Progress bar component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Progress bar component:

- progressbar (role)
- aria-valuemin (attribute)
- aria-valuemax (attribute)
- aria-valuenow (attribute)
- aria-label (attribute)

Keyboard interaction

The Progress bar component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Progress bar component.

| **Press** | **To do this** |

| --- | --- |

| **Tab** | **Moves the focus to the Progress bar element.** |

| **Ctrl + P** | **Prints the Progress bar.** |

Ensuring accessibility

The Progress bar component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Progress bar component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Progress bar component with accessibility tools.

See also

- [Accessibility in Syncfusion React components](#)

ProgressButton

Getting Started

This section explains how to create a simple ProgressButton and to configure it.

Dependencies

The list of dependencies required to use the ProgressButton component in your application is given as follows:

```
`js
|-- @syncfusion/ej2-react-splitbuttons
|-- @syncfusion/ej2-react-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
`,`
```

Installation and configuration

You can use [create-react-app](#) to setup the applications.

To install `create-react-app` run the following command.

```
`bash
npm install -g create-react-app
`,`
```

To set-up a React application in TypeScript environment, run the following command.

```
`bash
npx create-react-app my-app --template typescript
cd my-app
npm start
`,`
```

To set-up a React application in JavaScript environment, run the following command.

```
`bash
npx create-react-app my-app
cd my-app
```


npm start

,

Adding syncfusion packages

All the available Essential JS 2 packages are published in

[npmjs.com](https://www.npmjs.com) public registry.

You can choose the component that you want to install.

To install ProgressButton component, use the following command

```
`bash
```

```
npm install @syncfusion/ej2-react-splitbuttons --save
```

,

Adding CSS reference

Import the ProgressButton component required CSS references as follows in `src/App.css`.

```
`css
```

```
/ import the ProgressButton dependency styles /
```

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
```

,

Adding ProgressButton component

Now, you can start adding ProgressButton component in the application. For getting started, add the

ProgressButton component in `src/App.tsx` file using following code.

Add the below code in the `src/App.tsx` to initialize the ProgressButton.

```
`ts
```

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
```

```
import * as React from 'react';
```

```
import './App.css';
```

```
// To render ProgressButton.
```

```
function App() {
```

```
  return (<div style={{marginTop: '150px'}}>
```

```
    <ProgressButtonComponent content='Spin Left' />
```

```
  </div>
```

```
);
```

```
}
```

```
export default App;
```

```
,
```

Run the application

After completing the configuration required to render a basic ProgressButton, run the following command to

display the output in your default browser.

```
,
```

```
npm start
```

```
,
```

The following example shows a basic Progress button component.

APP.JSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render ProgressButton.
function App() {
  return (
    <div>
      <ProgressButtonComponent content='Spin Left' />
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

APP.TSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render ProgressButton.
function App() {
  return (
    <div>
      <ProgressButtonComponent content='Spin Left' />
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

ProgressButton supports different styles, types and sizes like [Button](#). In addition, it also supports **top** and **bottom** icon positions.

See Also

- [Spinner and Progress options](#)

```
<!-- markdownlint-disable MD002 MD022 -->
```

Spinner and progress in React Progress button component

Change spinner position

Spinner position can be changed by modifying the [position](#) property of [spinSettingsModel](#). By default, the spinner is positioned at the left of the ProgressButton. You can position it at the [left](#), [right](#), [top](#), [bottom](#), or [center](#) of the text content.

Change spinner size

Spinner size can be changed by modifying the [width](#) property of [spinSettingsModel](#). In this demo, the [width](#) is set to [20](#) to change the spinner size.

Spinner template

You can use custom spinner by specifying the [template](#) property of [spinSettingsModel](#) with custom styles.

The following sample demonstrates the above functionalities of the spinner.

APP.JSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let spinSettings = { position: 'Right', width: 20, template: '<div class="template"></div>' };
    return (<ProgressButtonComponent content='Submit' spinSettings={spinSettings}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

APP.TSX

```
import { ProgressButtonComponent, SpinSettingsModel } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let spinSettings : SpinSettingsModel = { position: 'Right', width: 20, template: '<div class="template"></div>' };
    return (
        <ProgressButtonComponent content='Submit' spinSettings={spinSettings}/>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

Progress

Content animation

The [content](#) of the ProgressButton can be animated during progress using the [effect](#) property of [animationSettingsModel](#). You can also set custom duration and timing function using the [duration](#) and

[easing](#) properties. The possible [effect](#) values are [None](#), [SlideLeft](#), [SlideRight](#), [SlideUp](#), [SlideDown](#), [ZoomIn](#), and [ZoomOut](#).

APP.JSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let animationSettings = { effect: 'SlideLeft', duration: 500, easing: 'linear' };
    let spinSettings = { position: 'Center' };
    return (<ProgressButtonComponent content='Slide Left'
enableProgress={true} animationSettings={animationSettings}
spinSettings={spinSettings}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

APP.TSX

```
import { AnimationSettingsModel, ProgressButtonComponent, SpinSettingsModel } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let animationSettings: AnimationSettingsModel = { effect: 'SlideLeft', duration: 500, easing: 'linear' };
    let spinSettings : SpinSettingsModel = { position: 'Center' };
    return (
        <ProgressButtonComponent content='Slide Left' enableProgress = {true}
animationSettings={animationSettings} spinSettings={spinSettings}/>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

Change step of the ProgressButton

The progress can be visualized at the specified interval by changing the [step](#) property in the [begin](#) event of the ProgressButton. In this demo, the [step](#) property is set to [20](#) to show progress at every 20% increment.

APP.JSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    function begin(args) {
        args.step = 20;
    }
    return (<ProgressButtonComponent content='Progress Step'
enableProgress={true} begin={begin} cssClass='e-hide-spinner'/>);
}
```

```

}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));

```

APP.TSX

```

import { ProgressButtonComponent, ProgressEventArgs } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  function begin(args: ProgressEventArgs): void {
    args.step = 20;
  }
  return (
    <ProgressButtonComponent content='Progress Step' enableProgress = {true}
    begin={begin} cssClass='e-hide-spinner'/>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));

```

The class `e-hide-spinner` hides the spinner in the ProgressButton, For more information, see [hide spinner](#) section.

Change progress dynamically

The progress can be changed dynamically by modifying the `percent` property in the ProgressButton events. In this demo, on 40% completion of progress, the `percent` property is set to `90` to show dynamic change of the progress.

APP.JSX

```

import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import { useState } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  const [state, setState] = useState({
    content: 'Progress'
  });
  return (<ProgressButtonComponent content={state.content}
  enableProgress={true} duration={15000} begin={begin} progress={progress}
  end={end} cssClass='e-hide-spinner'/>);
  function begin(args) {
    setState({ content: 'Progress ' + args.percent + '%' });
  }
  function progress(args) {
    setState({ content: 'Progress ' + args.percent + '%' });
    if (args.percent === 40) {
      args.percent = 90;
    }
  }
  function end(args) {
    setState({ content: 'Progress ' + args.percent + '%' });
  }
}

```

```

    }
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById('progress-button'));

```

APP.TSX

```

import { ProgressButtonComponent, ProgressEventArgs } from '@syncfusion/ej2-react-splitbuttons';
import { useState } from "react";
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  const [state, setState] = useState({
    content: 'Progress'
  });
  return (
    <ProgressButtonComponent content={state.content} enableProgress =
    {true} duration={15000} begin={begin} progress={progress} end={end}
    cssClass='e-hide-spinner' />
  );
  function begin(args: ProgressEventArgs): void {
    setState({ content: 'Progress ' + args.percent + '%' });
  }
  function progress(args: ProgressEventArgs): void {
    setState({ content: 'Progress ' + args.percent + '%' });
    if (args.percent === 40) {
      args.percent = 90;
    }
  }
  function end(args: ProgressEventArgs): void {
    setState({ content: 'Progress ' + args.percent + '%' });
  }
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));

```

The method [dataBind](#) applies the property changes immediately to the component.

Start and stop methods

You can pause and resume the progress using the [stop](#) and [start](#) methods, respectively. In this demo, clicking the ProgressButton will pause and resume the progress.

APP.JSX

```

{% raw %}
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useState } from "react";
function App() {
  let progressBtn;
  const [state, setState] = useState({
    content: 'Download',
    iconCss: 'e-btn-sb-icon e-download'
  });

```

```

    });
    return (<ProgressButtonComponent content={state.content}
enableProgress={true} duration={4000} iconCss={state.iconCss} end={end}
cssClass='e-hide-spinner' onClick={clickHandler} ref={(scope) => {
progressBtn = scope; }}/>);
    function end() {
        setState({ content: 'Download', iconCss: 'e-btn-sb-icon e-download'
    });
    }
    function clickHandler() {
        if (state.content === 'Download') {
            setState({ content: 'Pause', iconCss: 'e-btn-sb-icon e-pause'
        });
        }
        else if (state.content === 'Pause') {
            setState({ content: 'Resume', iconCss: 'e-btn-sb-icon e-play'
        });
        }
        progressBtn.stop();
    }
        else if (state.content === 'Resume') {
            setState({ content: 'Pause', iconCss: 'e-btn-sb-icon e-pause'
        });
        }
        progressBtn.start();
    }
    }
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
{% endraw %}

```

APP.TSX

```

{% raw %}
import { ProgressButtonComponent } from '@syncfusion/ej2-react-
splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useState } from "react";
function App() {
    let progressBtn: ProgressButtonComponent;
    const [state, setState] = useState({
        content: 'Download',
        iconCss: 'e-btn-sb-icon e-download'
    });
    return (
        <ProgressButtonComponent content={state.content} enableProgress = {true}
duration={4000}
        iconCss={state.iconCss} end={end} cssClass='e-hide-spinner'
onClick={clickHandler} ref={(scope)=>{progressBtn = scope as
ProgressButtonComponent;}}/>
    );
    function end(): void {
        setState({ content: 'Download', iconCss: 'e-btn-sb-icon e-download' });
    }

    function clickHandler(): void {

```

```

    if (state.content === 'Download') {
      setState({ content: 'Pause', iconCss: 'e-btn-sb-icon e-pause' });
    }
    else if (state.content === 'Pause') {
      setState({ content: 'Resume', iconCss: 'e-btn-sb-icon e-play' });
      progressBtn.stop();
    }
    else if (state.content === 'Resume') {
      setState({ content: 'Pause', iconCss: 'e-btn-sb-icon e-pause' });
      progressBtn.start();
    }
  }
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
{% endraw %}

```

See Also

- [How to hide spinner](#)
- [Customize ProgressBar using cssClass](#)

Accessibility in React ProgressBar component

The ProgressBar component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the ProgressBar component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

| Screen Reader Support | |

| Right-To-Left Support | |

| Color Contrast | |

| Mobile Device Support | |

| Keyboard Navigation Support | |

| [Accessibility Checker](#) Validation | |


```
| Axe-core Accessibility Validation |  |
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>
<div> - All
features of the component meet the requirement.</div>
<div> - Some features of the component do not meet the requirement.</div>
<div> - The component does not meet the requirement.</div>
```

WAI-ARIA attributes

The ProgressBar component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the ProgressBar component:

Attributes	Purpose
---	---
aria-label	Provides an accessible name for the icon only ProgressBar.
aria-disabled	Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable.

Keyboard interaction

The ProgressBar component followed the [keyboard interaction] guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the ProgressBar component.

Press	To do this
---	---
Enter / Space	Starts the progress.

Ensuring accessibility

The ProgressBar component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the ProgressBar component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the ProgressBar component with accessibility tools.

See also

- [Accessibility in Syncfusion React components](#)

Style and appearance in React Progress button component

To modify the ProgressButton appearance, you need to override the default CSS of ProgressButton component. Please find the list of CSS classes and its corresponding section in ProgressButton. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

CSS Class | Purpose of Class

- |.e-progress-btn|To customize the progress button
- |.e-progress-btn: hover|To customize the progress button on hover
- |.e-progress-btn: focus|To customize the progress button on focus
- |.e-progress-btn .e-spinner-pane .e-spinner-inner svg .e-path-circle|To customize the progress button spinner

See also

- [How to showcase ProgressButton as a progress bar](#)

How To

Change the text content and styles of the progressbutton during progress in React Progress button component

You can change the text content and styles of the ProgressButton during progress by changing the text content and the [cssClass](#) property at the [begin](#) and [end](#) events.

APP.JSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useState } from "react";
function App() {
    const [state, setState] = useState({
        content: 'Upload',
        cssClass: 'e-hide-spinner'
    });
    function begin() {
        setState({ content: 'Uploading...', cssClass: 'e-hide-spinner e-info' });
    }
    function end() {
        setState({ content: 'Success', cssClass: 'e-hide-spinner e-success' });
    }
    setTimeout(() => {
        setState({ content: 'Upload', cssClass: 'e-hide-spinner' });
    }, 500);
    return (<ProgressButtonComponent content={state.content} enableProgress
cssClass={state.cssClass} duration={4000} begin={begin} end={end}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

APP.TSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useState } from 'react';
function App() {
    const [state, setState] = useState({
        content: 'Upload',
        cssClass: 'e-hide-spinner'
    });
    function begin(): void {
        setState({ content: 'Uploading...', cssClass: 'e-hide-spinner e-info' });
    }
    function end(): void {
        setState({ content: 'Success', cssClass: 'e-hide-spinner e-success' });
    };
    setTimeout(() => {
        setState({ content: 'Upload', cssClass: 'e-hide-spinner' });
    }, 500)
}
return (
    <ProgressButtonComponent content={state.content} enableProgress
    cssClass={state.cssClass} duration={4000} begin={begin} end={end}/>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

Customize progress using cssclass in React Progress button component

You can customize the background filler UI using the [cssClass](#) property.

- Adding `e-vertical` to `cssClass` shows vertical progress.
- Adding `e-progress-top` to `cssClass` shows progress at the top.

You can also show reverse progress by adding custom class to the [cssClass](#) property.

APP.JSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    return (<div>
        <ProgressButtonComponent content='Vertical Progress'
        enableProgress={true} cssClass='e-hide-spinner e-vertical' duration={4000}/>
        <ProgressButtonComponent content='Progress Top' enableProgress={true}
        cssClass='e-hide-spinner e-progress-top' duration={4000}/>
        <ProgressButtonComponent content='Reverse Progress'
        enableProgress={true} cssClass='e-hide-spinner e-reverse-progress'
        duration={4000}/></div>);
}
```

```
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

APP.TSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  return (<div>
    <ProgressButtonComponent content='Vertical Progress' enableProgress =
    {true} cssClass='e-hide-spinner e-vertical' duration={4000}/>
    <ProgressButtonComponent content='Progress Top' enableProgress = {true}
    cssClass='e-hide-spinner e-progress-top' duration={4000}/>
    <ProgressButtonComponent content='Reverse Progress' enableProgress =
    {true} cssClass='e-hide-spinner e-reverse-progress' duration={4000}/></div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

Hide spinner in React Progress button component

You can hide spinner in the ProgressButton by setting the `e-hide-spinner` property to [cssClass](#).

APP.JSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  return (<ProgressButtonComponent content='Progress'
  enableProgress={true} cssClass='e-hide-spinner'/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

APP.TSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  return (
    <ProgressButtonComponent content='Progress' enableProgress = {true}
    cssClass='e-hide-spinner'/>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

Enable progress in button in React Progress button component

You can enable the background filler UI by setting the [enableProgress](#) property to `true`.

APP.JSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render ProgressButton.
function App() {
    return (<div>
        <ProgressButtonComponent content='Spin Left' enableProgress={true}/>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

APP.TSX

```
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render ProgressButton.
function App() {
    return (
        <div>
            <ProgressButtonComponent content='Spin Left' enableProgress= {true}/>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
```

Trace events of progress button in React Progress button component

The ProgressButton component triggers events based on its actions. The events can be used as extension points to perform custom operations.

The events available in ProgressButton are [fail](#), [begin](#), [progress](#), and [end](#).

APP.JSX

```
{% raw %}
import { ProgressButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useState } from 'react';
function App() {
    const [state, setState] = useState({
        eventTrace: ''
    });
    function begin(args) {
        updateEventLog(args);
    }
}
```

```

    }
    function end(args) {
        updateEventLog(args);
    }
    function progress(args) {
        updateEventLog(args);
    }
    function fail(args) {
        updateEventLog(args);
    }
    function updateEventLog(args) {
        setState({ eventTrace: state.eventTrace + args.name + ' Event
triggered. <br />' });
    }
    function btnClick() {
        setState({ eventTrace: '' });
    }
    return (<div className='control-section'>
        <div className='progress-btn-section'>
            <ProgressButtonComponent content='Progress'
enableProgress={true} begin={begin} end={end} progress={progress}
fail={fail}/>
        </div>
        <div className='property-section'>
            <table id='propertyTable' title='Event trace'>
                <tbody>
                    <th>Event trace:-</th>
                    <tr>
                        <td dangerouslySetInnerHTML={{ __html:
state.eventTrace }}/>
                    </tr>
                </tbody>
            </table>
        </div>
        <ButtonComponent id='clear' cssClass='e-small' content='Clear'
onClick={btnClick}/>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
{% endraw %}

```

APP.TSX

```

{% raw %}
import { ProgressButtonComponent, ProgressEventArgs } from '@syncfusion/ej2-
react-splitbuttons';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useState } from "react";
function App() {
    const [state, setState] = useState({
        eventTrace: ''
    });
    function begin(args: ProgressEventArgs): void {

```

```

        updateEventLog(args);
    }
    function end(args: ProgressEventArgs): void {
        updateEventLog(args);
    }
    function progress(args: ProgressEventArgs): void {
        updateEventLog(args);
    }
    function fail(args: Event): void {
        updateEventLog(args);
    }
    function updateEventLog(args: any): void {
        setState({ eventTrace: state.eventTrace + args.name + ' Event
triggered. <br />' });
    }
    function btnClick(): void {
        setState({ eventTrace: '' });
    }
    return (
        <div className='control-section'>
            <div className='progress-btn-section'>
                <ProgressButtonComponent content='Progress' enableProgress =
{true} begin={begin} end={end} progress={progress} fail={fail}/>
            </div>
            <div className='property-section'>
                <table id='propertyTable' title='Event trace'>
                    <tbody>
                        <th>Event trace:-</th>
                        <tr>
                            <td dangerouslySetInnerHTML={{__html:
state.eventTrace}}/>
                        </tr>
                    </tbody>
                </table>
            </div>
            <ButtonComponent id='clear' cssClass='e-small' content='Clear'
onClick={btnClick}/>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('progress-button'));
{% enddraw %}

```

Query Builder

Getting Started

This section explains how to create and configure a simple [React Query Builder component](#).

Dependencies

The list of dependencies required to use the Query Builder component in your application is given below:

```
`javascript`
```

```
|-- @syncfusion/ej2-react-querybuilder
|-- @syncfusion/ej2-react-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-querybuilder
|-- @syncfusion/ej2-datamanager
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-calenders
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-popups
\
```

Installation and Configuration

You can use [Create-react-app](#) to setup the applications. To install `create-react-app` run the following command.

```
`bash
npm install -g create-react-app
\
```

To set-up a React application in TypeScript environment, run the following command.

```
`bash
npx create-react-app my-app --template typescript
cd my-app
npm start
\
```

To set-up a React application in JavaScript environment, run the following command.

```
`bash
npx create-react-app my-app
cd my-app
npm start
\
```

Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) public registry.

To install Query Builder component, use the following command.

```
`bash
npm install @syncfusion/ej2-react-querybuilder --save
\
```


Adding CSS Reference

Import the Button component's required CSS references as follows in `src/App.css`.

```
`css
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-querybuilder/styles/material.css";
`
```

Adding Query Builder component to the Application

To include the Query Builder component in your application import the `QueryBuilderComponent` from `ej2-react-querybuilder` package in `App.tsx`.

Add the Query Builder component in application as shown in below code example.

```
`ts
import { ColumnsModel, QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import './App.css';
function App() {
  let columnData: ColumnsModel[] = [
    { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
    { field: 'FirstName', label: 'FirstName', type: 'string' },
    { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type: 'boolean', values: ['Mr.', 'Mrs.'] },
    { field: 'Title', label: 'Title', type: 'string' },
    { field: 'HireDate', label: 'HireDate', type: 'date', format: 'dd/MM/yyyy' },
    { field: 'Country', label: 'Country', type: 'string' },
    { field: 'City', label: 'City', type: 'string' }
  ];
  return (
    <QueryBuilderComponent width='100%' columns={columnData}/>
  );
}
```

```
);
}

export default App;
`
```

Run the application

Run the application in the browser using the following command:

```
`
npm start
`
```

The following example shows a basic Query Builder component.

APP.JSX

```
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let columnData = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    return (<QueryBuilderComponent width='100%'
columns={columnData}></QueryBuilderComponent>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
```

APP.TSX

```
import { ColumnsModel, QueryBuilderComponent } from '@syncfusion/ej2-react-
querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let columnData: ColumnsModel[] = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number'},
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ]
```

```

    ];
    return (
      <QueryBuilderComponent width='100%' columns={columnData}>
    </QueryBuilderComponent>
    );
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

You can also explore our [React Query Builder example](#) that shows how to render the Query Builder in React.

Columns in React Query builder component

The column definitions are used as the [dataSource](#) schema in the Query Builder. This plays a vital role in rendering column values. The query builder operations such as create or delete conditions and create or delete group they are performed based on the column definitions. The [field](#) property of the columns is necessary to map the data source values in the query builder columns.

If the column field is not specified in the [dataSource](#), the column values will be empty.

Auto generation

The [columns](#) are automatically generated when the [columns](#) declaration is empty or undefined while initializing the query builder. All the columns in the [dataSource](#) are bound as the query builder columns.

APP.JSX

```

import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
  return (<QueryBuilderComponent width='100%'
dataSource={employeeData}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
  return (
    <QueryBuilderComponent width='100%' dataSource={employeeData}/>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

When columns are auto-generated, the column type will be determined from the first record of the [dataSource](#).

Labels

By default, the column label is displayed from the column [field](#) value. To override the default label, you have to define the [label](#) value.

Operators

The operator for a column can be defined in the [operators](#) property.

The available operators and its supported data types are:

Operators	Description	Supported Types
-----	-----	-----
startswith	Checks whether the value begins with the specified value.	String
endswith	Checks whether the value ends with the specified value.	String
contains	Checks whether the value contains the specified value.	String
equal	Checks whether the value is equal to the specified value.	String Number Date Boolean
notequal	Checks whether the value is not equal to the specified value.	String Number Date Boolean
greaterthan	Checks whether the value is greater than the specified value.	Date Number
greaterthanorequal	Checks whether a value is greater than or equal to the specified value.	Date Number
lessthan	Checks whether the value is less than the specified value.	Date Number
lessthanorequal	Checks whether the value is less than or equal to the specified value.	Date Number
between	Checks whether the value is between the two-specific value.	Date Number
notbetween	Checks whether the value is not between the two-specific value.	Date Number
in	Checks whether the value is one of the specific values.	String Number
notin	Checks whether the value is not in the specific values.	String Number

Step

The Query Builder allows you to set the step values to the number fields. So that you can easily access the numeric textbox. Use the [step](#) property, to set the step value for number values.

Format

The Query Builder formats date and number values. Use the [format](#) property to format date and number values.

APP.JSX

```
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
```

```
function App() {
  let columnData = [
    { field: 'EmployeeID', label: 'Employee ID', operators: [{ key:
'Equal', value: 'equal' },
    { key: 'Greater than', value: 'greaterthan' }, { key: 'Less
than', value: 'lessthan' }], step: 10, type: 'number' },
    { field: 'FirstName', label: 'First Name', type: 'string' },
    { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
    { field: 'Title', label: 'Title', type: 'string' },
    { field: 'HireDate', label: 'Hire Date', type: 'date', format:
'dd/MM/yyyy' },
    { field: 'Country', label: 'Country', type: 'string' },
    { field: 'City', label: 'City', type: 'string' }
  ];
  let importRules = {
    'condition': 'and',
    'rules': [{
      'field': 'EmployeeID',
      'label': 'Employee ID',
      'operator': 'equal',
      'type': 'number',
      'value': 1001
    },
    {
      'field': 'HireDate',
      'label': 'Hire Date',
      'operator': 'equal',
      'type': 'date',
      'value': '07/05/1991'
    },
    {
      'field': 'Title',
      'label': 'Title',
      'operator': 'equal',
      'type': 'string',
      'value': 'Sales Manager'
    }
  ]
};
  return (<QueryBuilderComponent width='100%' dataSource={employeeData}
columns={columnData} rule={importRules}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
```

APP.TSX

```
import { ColumnsModel, QueryBuilderComponent, RuleModel } from
'@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
  let columnData: ColumnsModel[] = [
```

```

    { field: 'EmployeeID', label: 'Employee ID', operators: [{ key:
'Equal', value: 'equal' },
    { key: 'Greater than', value: 'greaterthan' }, { key: 'Less
than', value: 'lessthan' }], step: 10, type: 'number'},
    { field: 'FirstName', label: 'First Name', type: 'string' },
    { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
    { field: 'Title', label: 'Title', type: 'string' },
    { field: 'HireDate', label: 'Hire Date', type: 'date', format:
'dd/MM/yyyy' },
    { field: 'Country', label: 'Country', type: 'string' },
    { field: 'City', label: 'City', type: 'string' }
  ];
  let importRules: RuleModel = {
    'condition': 'and',
    'rules': [{
      'field': 'EmployeeID',
      'label': 'Employee ID',
      'operator': 'equal',
      'type': 'number',
      'value': 1001
    },
    {
      'field': 'HireDate',
      'label': 'Hire Date',
      'operator': 'equal',
      'type': 'date',
      'value': '07/05/1991'
    },
    {
      'field': 'Title',
      'label': 'Title',
      'operator': 'equal',
      'type': 'string',
      'value': 'Sales Manager'
    }
  ]
};
  return (
    <QueryBuilderComponent width='100%' dataSource={employeeData}
columns={columnData}
rule={importRules} />
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Validations

Validation allows you to validate the conditions and it display errors for invalid fields while using the [validateFields](#) method. To enable validation in the query builder , set the allowValidation to true. Column fields are validated after setting [allowValidation](#) as to true. So, you should manually configure the validation for Operator and, Value fields through [validation](#).

APP.JSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';

```

```

import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
    let qryBldrObj;
    let columnData = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number',
validation: { isRequired: true } },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', format: 'dd/MM/yyyy', label: 'HireDate', type:
'date' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    function onClick() {
        qryBldrObj.validateFields();
    }
    return (<div>
        <QueryBuilderComponent width='100%' dataSource={employeeData}
columns={columnData} allowValidation={true} ref={(scope) => { qryBldrObj =
scope; }}/>
        <div className="e-qb-button">
            <ButtonComponent id="validatebtn" cssClass='e-primary'
content='Validate Fields' onClick={onClick}/>
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { ColumnsModel, QueryBuilderComponent } from '@syncfusion/ej2-react-
querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
    let qryBldrObj: QueryBuilderComponent;
    let columnData: ColumnsModel[] = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number',
validation: { isRequired: true } },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },

```

```

    { field: 'HireDate', format: 'dd/MM/yyyy', label: 'HireDate', type:
'date' },
    { field: 'Country', label: 'Country', type: 'string' },
    { field: 'City', label: 'City', type: 'string' }
  ];
  function onClick(): void {
    qryBldrObj.validateFields();
  }
  return (
    <div>
      <QueryBuilderComponent width='100%' dataSource={employeeData}
columns={columnData} allowValidation={true} ref={(scope) => { qryBldrObj =
scope as QueryBuilderComponent; }}/>
      <div className="e-qb-button">
        <ButtonComponent id="validatebtn" cssClass='e-primary'
content='Validate Fields' onClick = {onClick} />
      </div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% endraw %}

```

Set [isRequired](#) validation for Operator and Value fields.

Set [min](#), [max](#) values for number values.

Data binding in React Query builder component

The Query Builder uses **DataManager**, which supports both RESTful JSON data services binding and local JavaScript object array binding. The **dataSource** property can be assigned either with the instance of **DataManager** or JavaScript object array collection. It supports two kind of databinding method:

- Local data
- Remote data

Local data

To bind local data to the query builder, you can assign the [dataSource](#) property with a JavaScript object array. The local data source can also be provided as an instance of the **DataManager**.

APP.JSX

```

import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
  let columnData = [
    { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
    { field: 'FirstName', label: 'FirstName', type: 'string' },
    { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
    { field: 'Title', label: 'Title', type: 'string' },

```



```

    { field: 'HireDate', label: 'HireDate', type: 'date', format:
'dd/MM/yyyy' },
    { field: 'Country', label: 'Country', type: 'string' },
    { field: 'City', label: 'City', type: 'string' }
  ];
  let importRules = {
    'condition': 'and',
    'rules': [{
      'field': 'EmployeeID',
      'label': 'EmployeeID',
      'operator': 'equal',
      'type': 'number',
      'value': 1
    },
    {
      'field': 'Title',
      'label': 'Title',
      'operator': 'equal',
      'type': 'string',
      'value': 'Sales Manager'
    }
  ]
};
  return (<QueryBuilderComponent width='100%' dataSource={employeeData}
columns={columnData} rule={importRules}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, RuleModel } from
'@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
  let columnData: ColumnsModel[] = [
    { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
    { field: 'FirstName', label: 'FirstName', type: 'string' },
    { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
    { field: 'Title', label: 'Title', type: 'string' },
    { field: 'HireDate', label: 'HireDate', type: 'date', format:
'dd/MM/yyyy' },
    { field: 'Country', label: 'Country', type: 'string' },
    { field: 'City', label: 'City', type: 'string' }
  ];
  let importRules: RuleModel = {
    'condition': 'and',
    'rules': [{
      'field': 'EmployeeID',
      'label': 'EmployeeID',
      'operator': 'equal',
      'type': 'number',
      'value': 1
    }
  ]
};

```

```

    },
    {
      'field': 'Title',
      'label': 'Title',
      'operator': 'equal',
      'type': 'string',
      'value': 'Sales Manager'
    }
  ]
};
return (
  <QueryBuilderComponent width='100%' dataSource={employeeData}
  columns={columnData} rule={importRules} />
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

By default, **DataManager** uses **JsonAdaptor** for local data-binding.

Remote data

To bind remote data to the query builder, assign service data as an instance of **DataManager** to the [dataSource](#) property. To interact with remote data source, provide the endpoint [url](#).

APP.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
function App() {
  let data = new DataManager({
    url:
    'https://services.odata.org/v4/Northwind/Northwind.svc/Employees/',
    adaptor: new ODataV4Adaptor(),
  });
  let columnData = [
    { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
    { field: 'FirstName', label: 'FirstName', type: 'string' },
    { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
    'boolean', values: ['Mr.', 'Mrs.'] },
    { field: 'Title', label: 'Title', type: 'string' },
    { field: 'HireDate', label: 'HireDate', type: 'date', format:
    'dd/MM/yyyy' },
    { field: 'Country', label: 'Country', type: 'string' },
    { field: 'City', label: 'City', type: 'string' }
  ];
  let importRules = {
    'condition': 'and',
    'rules': [{
      'label': 'EmployeeID',
      'field': 'EmployeeID',
      'type': 'number',
      'operator': 'equal',
      'value': 1
    }
  ],
  {

```

```

        'label': 'Title',
        'field': 'Title',
        'type': 'string',
        'operator': 'equal',
        'value': 'Sales Manager'
    }]
    };
    return (<QueryBuilderComponent width='100%' dataSource={data}
columns={columnData} rule={importRules}></QueryBuilderComponent>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { QueryBuilderComponent, ColumnsModel, RuleModel } from
'@syncfusion/ej2-react-querybuilder';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
function App() {
    let data = new DataManager({
        url:
'https://services.odata.org/v4/Northwind/Northwind.svc/Employees/',
        adaptor: new ODataV4Adaptor()
    });
    let columnData: ColumnsModel[] = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules: RuleModel = {
        'condition': 'and',
        'rules': [{
            'label': 'EmployeeID',
            'field': 'EmployeeID',
            'type': 'number',
            'operator': 'equal',
            'value': 1
        },
        {
            'label': 'Title',
            'field': 'Title',
            'type': 'string',
            'operator': 'equal',
            'value': 'Sales Manager'
        }
    ]
    };
    return (

```

```

        <QueryBuilderComponent width='100%' dataSource={data}
        columns={columnData} rule={importRules} ></QueryBuilderComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

By default, **DataManager** uses **ODataAdaptor** for remote data-binding.

Binding with OData services

OData is a standardized protocol for creating and consuming data. You can retrieve data from OData service using the DataManager. Refer to the following code example for remote Data binding using OData service.

APP.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import { DataManager, ODataAdaptor } from '@syncfusion/ej2-data';
function App() {
    let data = new DataManager({
        url: 'https://services.syncfusion.com/js/production/api/orders/',
        adaptor: new ODataAdaptor,
        crossDomain: true
    });
    let columnData = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
        'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
        'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules = {
        'condition': 'and',
        'rules': [{
            'field': 'EmployeeID',
            'label': 'EmployeeID',
            'operator': 'equal',
            'type': 'number',
            'value': 1
        },
        {
            'field': 'Title',
            'label': 'Title',
            'operator': 'equal',
            'type': 'string',
            'value': 'Sales Manager'
        }
    ]
    };
    return (<QueryBuilderComponent width='100%' dataSource={data}
    columns={columnData} rule={importRules}></QueryBuilderComponent>);
}

```

```

}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { QueryBuilderComponent, ColumnsModel, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import { DataManager, ODataAdaptor } from '@syncfusion/ej2-data';
function App() {
    let data = new DataManager({
        url: 'https://services.syncfusion.com/js/production/api/orders/',
        adaptor: new ODataAdaptor,
        crossDomain: true
    });
    let columnData: ColumnsModel[] = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules: RuleModel = {
        'condition': 'and',
        'rules': [{
            'field': 'EmployeeID',
            'label': 'EmployeeID',
            'operator': 'equal',
            'type': 'number',
            'value': 1
        },
        {
            'field': 'Title',
            'label': 'Title',
            'operator': 'equal',
            'type': 'string',
            'value': 'Sales Manager'
        }
    ]
    };
    return (
        <QueryBuilderComponent width='100%' dataSource={data}
columns={columnData} rule={importRules} ></QueryBuilderComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Binding with OData v4 services

The ODataV4 is an improved version of OData protocols, and the **DataManager** can also retrieve and consume OData v4 services. For more details on OData v4 services, refer to the [odata documentation](#). To bind OData v4 service, use the **ODataV4Adaptor**.

APP.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
function App() {
    let data = new DataManager({
        url:
        'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
        adaptor: new ODataV4Adaptor
    });
    let columnData = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
        'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
        'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules = {
        'condition': 'and',
        'rules': [{
            'label': 'EmployeeID',
            'field': 'EmployeeID',
            'type': 'number',
            'operator': 'equal',
            'value': 1
        },
        {
            'label': 'Title',
            'field': 'Title',
            'type': 'string',
            'operator': 'equal',
            'value': 'Sales Manager'
        }
    ]
    };
    return (<QueryBuilderComponent width='100%' dataSource={data}
    columns={columnData} rule={importRules}></QueryBuilderComponent>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
```

APP.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
```

```

import { QueryBuilderComponent, ColumnsModel, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
function App() {
    let data = new DataManager({
        url:
        'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
        adaptor: new ODataV4Adaptor
    });
    let columnData: ColumnsModel[] = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
        'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
        'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules: RuleModel = {
        'condition': 'and',
        'rules': [{
            'label': 'EmployeeID',
            'field': 'EmployeeID',
            'type': 'number',
            'operator': 'equal',
            'value': 1
        },
        {
            'label': 'Title',
            'field': 'Title',
            'type': 'string',
            'operator': 'equal',
            'value': 'Sales Manager'
        }
    ]
    };
    return (
        <QueryBuilderComponent width='100%' dataSource={data}
        columns={columnData} rule={importRules} ></QueryBuilderComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Web API

You can use **WebApiAdaptor** to bind query builder with Web API created using OData endpoint.

`ts

```
import * as React from 'react';
```

```
import * as ReactDOM from 'react-dom';
```

```
import { QueryBuilderComponent, ColumnsModel, RuleModel } from '@syncfusion/ej2-react-
querybuilder';
```

```
function App() {
  let data = new DataManager({
    url: '/api/OrderAPI',
    adaptor: new WebApiAdaptor
  });
  let columnData: ColumnsModel[] = [
    { field: 'EmployeeID', label: 'EmployeeID', type: 'number'},
    { field: 'FirstName', label: 'FirstName', type: 'string' },
    { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type: 'boolean', values: ['Mr.', 'Mrs.'], },
    { field: 'Title', label: 'Title', type: 'string' },
    { field: 'HireDate', label: 'HireDate', type: 'date', format: 'dd/MM/yyyy' },
    { field: 'Country', label: 'Country', type: 'string' },
    { field: 'City', label: 'City', type: 'string' }
  ];
  let importRules: RuleModel = {
    'condition': 'and',
    'rules': [{
      'label': 'EmployeeID',
      'field': 'EmployeeID',
      'type': 'number',
      'operator': 'equal',
      'value': 1
    },
    {
      'label': 'Title',
      'field': 'Title',
      'type': 'string',
      'operator': 'equal',
      'value': 'Sales Manager'
    }
  ]
};
  return (
```



```
<QueryBuilderComponent width='100%' dataSource={data} columns={columnData} rule={importRules}
></QueryBuilderComponent>

);
}

export default App;

ReactDOM.render(<App />,document.getElementById('querybuilder'));
`ts

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
function App() {
let data = new DataManager({
url: '/api/OrderAPI',
adaptor: new WebApiAdaptor
});
let columnData = [
{ field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
{ field: 'FirstName', label: 'FirstName', type: 'string' },
{ field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type: 'boolean', values: ['Mr.', 'Mrs.'] },
{ field: 'Title', label: 'Title', type: 'string' },
{ field: 'HireDate', label: 'HireDate', type: 'date', format: 'dd/MM/yyyy' },
{ field: 'Country', label: 'Country', type: 'string' },
{ field: 'City', label: 'City', type: 'string' }
];
let importRules = {
'condition': 'and',
'rules': [{
'label': 'EmployeeID',
'field': 'EmployeeID',
'type': 'number',
'operator': 'equal',
'value': 1
```

```

},
{
  'label': 'Title',
  'field': 'Title',
  'type': 'string',
  'operator': 'equal',
  'value': 'Sales Manager'
}]
};

return (<QueryBuilderComponent width='100%' dataSource={data} columns={columnData}
rule={importRules}></QueryBuilderComponent>);
}

export default App;

ReactDOM.render(<App />, document.getElementById('querybuilder'));
`

```

Data Manager

You can use the created conditions in DataManager through the `getPredicate` method. This method creates predicates which is used as conditions in DataManager. In this example given below, `getValidRules` method is used to get the valid queried data.

APP.JSX

```

{% raw %}
import { DataManager, Query } from '@syncfusion/ej2-data';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import { useState } from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
  let qryBldrObj;
  let columnData = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  let importRules = {
    'condition': 'or',
    'rules': [{
      'field': 'TaskID',
      'label': 'Task ID',

```

```

        'operator': 'equal',
        'type': 'number',
        'value': 1
    ]
    });
    const [state, setState] = useState({
        result: [],
        style: { display: 'none' }
    });
    function onClick() {
        const validRule = qryBldrObj.getValidRules(qryBldrObj.rule);
        const dataManagerQuery = new Query().select(['TaskID', 'Category',
'Status']).where(qryBldrObj.getPredicate(validRule)).take(8);
        setState({ result: new
DataManager(hardwareData).executeLocal(dataManagerQuery), style: { display:
'block' } });
    }
    return (<div>
        <QueryBuilderComponent width='100%' dataSource={hardwareData}
columns={columnData} rule={importRules} ref={(scope) => { qryBldrObj =
scope; }}/>
        <div className="e-qb-button">
            <ButtonComponent id="getdata" cssClass='e-primary'
content='get data' onClick={onClick}/>
        </div>
        <table id="datatable" className="e-table" style={state.style}>
            <thead>
                <tr><th>TaskID</th><th>Category</th><th>Status</th></tr>
            </thead>
            <tbody>{state.result.map(function func(item, key) {
                return (<tr key={key}>
                    <td>{item.TaskID}</td>
                    <td>{item.Category}</td>
                    <td>{item.Status}</td>
                </tr>);
            })}</tbody>
        </table>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% endraw %}

```

APP.TSX

```

{% raw %}
import { DataManager, Query } from '@syncfusion/ej2-data';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { ColumnsModel, QueryBuilderComponent, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import { useState } from "react";
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {

```

```

let qryBldrObj: QueryBuilderComponent;
let columnData: ColumnsModel[] = [
  { field: 'TaskID', label: 'Task ID', type: 'number' },
  { field: 'Name', label: 'Name', type: 'string' },
  { field: 'Category', label: 'Category', type: 'string' },
  { field: 'SerialNo', label: 'Serial No', type: 'string' },
  { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
  { field: 'Status', label: 'Status', type: 'string' }
];
let importRules: RuleModel = {
  'condition': 'or',
  'rules': [{
    'field': 'TaskID',
    'label': 'Task ID',
    'operator': 'equal',
    'type': 'number',
    'value': 1
  }]
};
const [state, setState] = useState({
  result: [],
  style: {display: 'none'}
});
function onClick(): void {
  const validRule = qryBldrObj.getValidRules(qryBldrObj.rule);
  const dataManagerQuery = new Query().select(['TaskID', 'Category', 'Status']).where(qryBldrObj.getPredicate(validRule)).take(8);
  setState({result: new
DataManager(hardwareData).executeLocal(dataManagerQuery), style: {display: 'block'}});
}
return (
  <div>
    <QueryBuilderComponent width='100%' dataSource={hardwareData}
columns={columnData} rule={importRules}
      ref={(scope) => { qryBldrObj = scope as
QueryBuilderComponent; }}/>
    <div className="e-qb-button">
      <ButtonComponent id="getdata" cssClass='e-primary'
content='get data' onClick = {onClick} />
    </div>
    <table id="datatable" className="e-table" style={state.style}>
      <thead>
        <tr><th>TaskID</th><th>Category</th><th>Status</th></tr>
      </thead>
      <tbody>{state.result.map(function func(item, key) {
        return (
          <tr key = {key}>
            <td>{item.TaskID}</td>
            <td>{item.Category}</td>
            <td>{item.Status}</td>
          </tr>
        )
      })}</tbody>
    </table>
  </div>
);

```

```

}
export default App;
ReactDOM.render(<App />,document.getElementById('querybuilder'));
{% endraw %}

```

Complex Data Binding

Complex Data Binding allows you to create subfield for columns. To implement complex data binding, either bind the complex data in nested columns or specify complex data source and separator must be given in querybuilder.

In the following sample, complex data was bound in nested columns.

APP.JSX

```

{% raw %}
import { QueryBuilderComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-react-querybuilder';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let qryBldrObj;
  let importRules = {
    condition: 'and',
    rules: [{
      label: 'ID',
      field: 'Employee.ID',
      type: 'string',
      operator: 'equal',
      value: 0
    },
    {
      label: 'Last Name',
      field: 'Name.LastName',
      type: 'string',
      operator: 'contains',
      value: 'malan'
    },
    {
      condition: 'or',
      rules: [{
        label: 'City',
        field: 'Country.State.City',
        operator: 'startswith',
        type: 'string',
        value: 'U'
      },
      {
        label: 'Region',
        field: 'Country.Region',
        operator: 'endswith',
        type: 'string',
        value: 'C'
      },
      {
        label: 'Name',

```

```

        field: 'Country.Name',
        operator: 'isnotempty'
    }
  ]],
  });
  let columns1 = [
    { field: 'ID', label: 'ID', type: 'number' },
    { field: 'DOB', label: 'Date of birth', type: 'date' },
    { field: 'HireDate', label: 'Hire Date', type: 'date' },
    { field: 'Salary', label: 'Salary', type: 'number' },
    { field: 'Age', label: 'Age', type: 'number' },
    { field: 'Title', label: 'Title', type: 'string' }
  ];
  let columns2 = [
    { field: 'FirstName', label: 'First Name', type: 'string' },
    { field: 'LastName', label: 'Last Name', type: 'string' }
  ];
  let columns3 = [
    { field: 'State', label: 'State', columns: [
      { field: 'City', label: 'City', type: 'string' },
      { field: 'Zipcode', label: 'Zip Code', type: 'number' }
    ] },
    { field: 'Region', label: 'Region', type: 'string' },
    { field: 'Name', label: 'Name', type: 'string' }
  ];
  function onReset() {
    qryBldrObj.reset();
  }
  function onSetsqlRules() {
    qryBldrObj.setRulesFromSql("Employee.ID = 0 AND Name.LastName LIKE ('%malan%') AND (Country.State.City LIKE ('U%') AND Country.Region LIKE ('%c') AND Country.Name IS NOT EMPTY)");
  }
  function onSetrules() {
    qryBldrObj.setRules(importRules);
  }
  return (<div>
    <div>
      <table>
        <tr>
          <td> <ButtonComponent id="reset" className="e-control e-danger e-btn e-small" onClick={onReset}>Reset</ButtonComponent>
          <td> <ButtonComponent id="rule" className="e-control e-success e-btn e-small"
            onClick={onSetsqlRules}>SetSqlRules</ButtonComponent>
          <td> <ButtonComponent id="sql" className="e-control e-success e-btn e-small"
            onClick={onSetrules}>SetRules</ButtonComponent>
        </tr>
      </table>
    </div>
    <QueryBuilderComponent width="100%" rule={importRules}
      id='querybuilder' separator="." enableNotCondition="true" ref={(scope) => {
        qryBldrObj = scope; }}>
      <ColumnsDirective>

```

```

        <ColumnDirective field="Employee" label="Employee"
columns={columns1}/>
        <ColumnDirective field="Name" label="Name"
columns={columns2}/>
        <ColumnDirective field="Country" label="Country"
columns={columns3}/>
    </ColumnsDirective>
</QueryBuilderComponent>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder-component'));
{% endraw %}

```

APP.TSX

```

{% raw %}
import { QueryBuilderComponent, ColumnsDirective, ColumnDirective,
RuleChangeEventArgs } from '@syncfusion/ej2-react-querybuilder';
import { RuleModel, ColumnsModel } from '@syncfusion/ej2-querybuilder';
import { RadioButtonComponent, ButtonComponent, ChangeEventArgs } from
 '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    let qryBldrObj: QueryBuilderComponent;
    let importRules: RuleModel = {
        condition: 'and',
        rules: [{
            label: 'ID',
            field: 'Employee.ID',
            type: 'string',
            operator: 'equal',
            value: 0
        },
        {
            label: 'Last Name',
            field: 'Name.LastName',
            type: 'string',
            operator: 'contains',
            value: 'malan'
        },
        {
            condition: 'or',
            rules: [{
                label: 'City',
                field: 'Country.State.City',
                operator: 'startswith',
                type: 'string',
                value: 'U'
            },
            {
                label: 'Region',
                field: 'Country.Region',
                operator: 'endswith',
                type: 'string',

```

```

        value: 'c'
      },
      {
        label: 'Name',
        field: 'Country.Name',
        operator: 'isnotempty'
      }
    ]
  ]],
};
let columns1: ColumnsModel[] = [
  { field: 'ID', label: 'ID', type: 'number' },
  { field: 'DOB', label: 'Date of birth', type: 'date' },
  { field: 'HireDate', label: 'Hire Date', type: 'date' },
  { field: 'Salary', label: 'Salary', type: 'number' },
  { field: 'Age', label: 'Age', type: 'number' },
  { field: 'Title', label: 'Title', type: 'string' }
];
let columns2: ColumnsModel[] = [
  { field: 'FirstName', label: 'First Name', type: 'string' },
  { field: 'LastName', label: 'Last Name', type: 'string' }
];
let columns3: ColumnsModel[] = [
  { field: 'State', label: 'State', columns: [
    { field: 'City', label: 'City', type: 'string' },
    { field: 'Zipcode', label: 'Zip Code', type: 'number' } ] },
  { field: 'Region', label: 'Region', type: 'string' },
  { field: 'Name', label: 'Name', type: 'string' }
];
function onReset(): void{
  qryBldrObj.reset();
}
function onSetsqlRules(): void{
  qryBldrObj.setRulesFromSql("Employee.ID = 0 AND Name.LastName
LIKE ('%malan%') AND (Country.State.City LIKE ('U%') AND Country.Region LIKE
('%c') AND Country.Name IS NOT EMPTY)");
}
function onSetrules(): void{
  qryBldrObj.setRules(importRules);
}
return (


|                                                                                                                    |                                                                                                                                |                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <ButtonComponent id="reset" className="e-control e-danger e-btn e-small" onClick={onReset}>Reset</ButtonComponent> | <ButtonComponent id="rule" className="e-control e-success e-btn e-small" onClick={onSetsqlRules}>SetSqlRules</ButtonComponent> | <ButtonComponent id="sql" className="e-control e-success e-btn e-small" onClick={onSetrules}>SetRules</ButtonComponent> |
|--------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|


```



```

        <QueryBuilderComponent width='100%' rule={importRules}
id='querybuilder' separator= "." enableNotCondition ="true" ref={(scope) =>
{ qryBldrObj = scope; }} >
            <ColumnsDirective>
                <ColumnDirective field="Employee" label="Employee"
columns={columns1} />
                <ColumnDirective field="Name" label="Name"
columns={columns2} />
                <ColumnDirective field="Country" label="Country"
columns={columns3} />
            </ColumnsDirective>
        </QueryBuilderComponent>
    </div>;
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder-component'));
{% endraw %}

```

Filtering in React Query builder component

Query Builder allows you to create or delete conditions and groups. You can use [showButtons](#) to enable/disable these buttons.

You can create or delete conditions by interacting through the user interface and methods.

- Use the [addRules](#), and [deleteRules](#) methods to create/delete conditions.
- Use [addGroups](#), and [deleteGroups](#) methods to create/delete groups.

APP.JSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
    let qryBldrObj;
    let ButtonOptions = {
        groupDelete: true,
        groupInsert: true,
        ruleDelete: true
    };
    let columnData = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'First Name', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules = {

```

```

        'condition': 'and',
        'rules': [{
            'field': 'EmployeeID',
            'label': 'EmployeeID',
            'operator': 'equal',
            'type': 'number',
            'value': 1001
        },
        {
            'field': 'Title',
            'label': 'Title',
            'operator': 'equal',
            'type': 'string',
            'value': 'Sales Manager'
        }
    ]
    };
    function addGroup() {
        qryBldrObj.addGroups([{ 'condition': 'and', 'rules': [{ 'label':
'First Name', 'field': 'FirstName', 'type': 'string', 'operator':
'startswith', 'value': 'v' }] }], 'group0');
    }
    function addRule() {
        qryBldrObj.addRules([{ 'label': 'City', 'field': 'City', 'type':
'string', 'operator': 'equal', 'value': 'US' }], 'group0');
    }
    function deleteGroup() {
        qryBldrObj.deleteGroups(['group1']);
    }
    return (<div>
        <QueryBuilderComponent width='100%' dataSource={employeeData}
columns={columnData} rule={importRules} ref={(scope) => { qryBldrObj =
scope; }} showButtons={ButtonOptions}/>
        <div className="e-qb-button">
            <ButtonComponent id="addgroup" cssClass='e-primary'
content='Add Group' onClick={addGroup}/>
            <ButtonComponent id="addrules" cssClass='e-primary'
content='Add Rule' onClick={addRule}/>
            <ButtonComponent id="deletegroups" cssClass='e-primary'
content='Delete Group' onClick={deleteGroup}/>
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { ColumnsModel, QueryBuilderComponent, RuleModel, ShowButtonsModel }
from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';

```

```

function App() {
  let qryBldrObj: QueryBuilderComponent;
  let ButtonOptions: ShowButtonsModel = {
    groupDelete: true,
    groupInsert: true,
    ruleDelete: true
  };
  let columnData: ColumnsModel[] = [
    { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
    { field: 'FirstName', label: 'First Name', type: 'string' },
    { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
    { field: 'Title', label: 'Title', type: 'string' },
    { field: 'HireDate', label: 'HireDate', type: 'date', format:
'dd/MM/yyyy' },
    { field: 'Country', label: 'Country', type: 'string' },
    { field: 'City', label: 'City', type: 'string' }
  ];
  let importRules: RuleModel = {
    'condition': 'and',
    'rules': [{
      'field': 'EmployeeID',
      'label': 'EmployeeID',
      'operator': 'equal',
      'type': 'number',
      'value': 1001
    },
    {
      'field': 'Title',
      'label': 'Title',
      'operator': 'equal',
      'type': 'string',
      'value': 'Sales Manager'
    }
  ]
  };
  function addGroup(): void {
    qryBldrObj.addGroups([{'condition': 'and', 'rules': [{'label': 'First
Name', 'field': 'FirstName', 'type': 'string', 'operator':
'startswith', 'value': 'v' }]}], 'group0');
  }
  function addRule(): void {
    qryBldrObj.addRules([{'label': 'City', 'field': 'City', 'type':
'string', 'operator': 'equal', 'value': 'US'}], 'group0');
  }
  function deleteGroup(): void {
    qryBldrObj.deleteGroups(['group1']);
  }
  return (
    <div>
      <QueryBuilderComponent width='100%' dataSource={employeeData}
columns={columnData}
      rule={importRules} ref={(scope) => { qryBldrObj = scope as
QueryBuilderComponent; }} showButtons={ButtonOptions}/>
      <div className="e-qb-button">
        <ButtonComponent id="addgroup" cssClass='e-primary'
content='Add Group' onClick = {addGroup}/>

```

```

        <ButtonComponent id="addrules" cssClass='e-primary'
content='Add Rule' onClick = {addRule}/>
        <ButtonComponent id="deletegroups" cssClass='e-primary'
content='Delete Group' onClick = {deleteGroup}/>
    </div>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% enddraw %}

```

Templates in React Query builder component

Templates allows users to define customized header and own user interface for columns.

Header Template

Header Template allows to define your own user interface for Header, which includes creating or deleting rules and groups and to customize the AND/OR condition and NOT condition options. To implement header template, you can create the user interface as React component and assign the values when requestType is header-template-create in `actionBegin` event.

In the following sample dropdown, splitbutton and button are used as the custom components in the header.

APP.JSX

```

import { QueryBuilderComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { HeaderTemplate } from './template';
function App() {
    let qryBldrObj;
    let importRules = {
        'condition': 'and', 'not': true,
        'rules': [{
            'field': 'Age',
            'label': 'Age',
            'operator': 'equal',
            'type': 'number',
            'value': 30
        },
        {
            'label': 'LastName',
            'field': 'LastName',
            'type': 'string',
            'operator': 'equal',
            'value': 'vinit'
        },
        {
            'condition': 'or',
            'rules': [{
                'label': 'Age',
                'field': 'Age',
                'type': 'number',

```

```

                'operator': 'equal',
                'value': 34
            }
        ]
    };
    function headerTemplate(props) {
        return (<HeaderTemplate {...props}/>);
    }
    return (<div>
        <QueryBuilderComponent width='100%' rule={importRules}
headerTemplate={headerTemplate} id='querybuilder' enableNotCondition="true">
            <ColumnsDirective>
                <ColumnDirective field="EmployeeID" label="Employee ID"
type="number"/>
                <ColumnDirective field="LastName" label="Last Name"
type="string"/>
                <ColumnDirective field="FirstName" label="First Name"
type="string"/>
                <ColumnDirective field="Age" label="Age" type="number"/>
                <ColumnDirective field="City" label="City" type="string"/>
                <ColumnDirective field="Country" label="Country"
type="string"/>
            </ColumnsDirective>
        </QueryBuilderComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder-component'));

```

APP.TSX

```

import { QueryBuilderComponent, ColumnsDirective, ColumnDirective } from
'@syncfusion/ej2-react-querybuilder';
import { RuleModel } from '@syncfusion/ej2-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { HeaderTemplate } from './template';
function App() {
    let qryBldrObj: QueryBuilderComponent;
    let importRules: RuleModel = {
        'condition': 'and', 'not': true,
        'rules': [{
            'field': 'Age',
            'label': 'Age',
            'operator': 'equal',
            'type': 'number',
            'value': 30
        },
        {
            'label': 'LastName',
            'field': 'LastName',
            'type': 'string',
            'operator': 'equal',
            'value': 'vinit'
        }
    ],
}

```

```

        {
            'condition': 'or',
            'rules': [{
                'label': 'Age',
                'field': 'Age',
                'type': 'number',
                'operator': 'equal',
                'value': 34
            }]
        }
    ]
};
function headerTemplate(props) {
    return (<HeaderTemplate {...props}/>);
}
return (<div>
    <QueryBuilderComponent width='100%' rule={importRules}
headerTemplate= {headerTemplate} id='querybuilder' enableNotCondition
="true" >
        <ColumnsDirective>
            <ColumnDirective field="EmployeeID" label="Employee ID"
type="number"/>
            <ColumnDirective field="LastName" label="Last Name"
type="string"/>
            <ColumnDirective field="FirstName" label="First Name"
type="string"/>
            <ColumnDirective field="Age" label="Age" type="number"/>
            <ColumnDirective field="City" label="City" type="string"/>
            <ColumnDirective field="Country" label="Country"
type="string"/>
        </ColumnsDirective>
    </QueryBuilderComponent>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder-component'));

```

TEMPLATE.JSX

```

import * as React from 'react';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { getComponent, closest } from '@syncfusion/ej2-base';
import { CheckBoxComponent, ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DropDownButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
export function HeaderTemplate(props) {
    let ds = [{ 'key': 'AND', 'value': 'and' }, { 'key': 'OR', 'value': 'or' }
    ]];
    let qryBldrObj;
    let fields = { text: 'key', value: 'value' };
    let ddbitems = [
        {
            text: 'AddGroup',
            iconCss: 'e-icons e-add-icon e-addgroup'
        },
    ],

```

```

    {
      text: 'AddCondition',
      iconCss: 'e-icons e-add-icon e-addrule'
    }
  ];
  let state = Object.assign({}, props);
  qryBldrObj = getComponent(document.getElementById('querybuilder'),
'query-builder');
  function onChange(args) {
    qryBldrObj.notifyChange(args.checked, args.event.target, 'not');
  }
  function conditionChange(args) {
    qryBldrObj.notifyChange(args.value, args.element, 'condition');
  }
  function onSelect(event) {
    let addbtn = closest(event.element, '.e-dropdown-popup');
    let ddbId = addbtn.id;
    let ddb = ddbId.split('_');
    if (event.item.text === 'AddGroup') {
      qryBldrObj.addGroups([ condition: 'or', 'rules': [{}], not:
false ], ddb[1]);
    }
    else if (event.item.text === 'AddCondition') {
      qryBldrObj.addRules([{}], ddb[1]);
    }
  }
  function onClick(args) {
    qryBldrObj.deleteGroup(closest(args.target.offsetParent, '.e-group-
container'));
  }
  const args = state;
  return (<div className="e-groupheader">
    {(() => {
      if (args.notCondition !== undefined) {
        return (<button className="e-cb-wrapper">
          <CheckBoxComponent id={args.ruleID + "_notOption"}
label="not" checked={args.notCondition} change={onChange}/>
        </button>);
      }
    })()}
    <DropDownListComponent id={args.ruleID + "_cndtn"} cssClass='e-
custom-group-btn' dataSource={ds} fields={fields} value={args.condition}
change={conditionChange}/>
    <DropDownButtonComponent id={args.ruleID + "_addbtn"}
items={ddbitems} cssClass="e-round e-small e-caret-hide e-addrulegroup e-
add-btn" iconCss="e-icons e-add-icon"
select={onSelect}></DropDownButtonComponent>
    {(() => {
      if (args.ruleID !== "querybuilder_group0") {
        return (<ButtonComponent id={args.ruleID + "_dltbtn"}
cssClass='e-btn e-delete-btn e-lib e-small e-round e-icon-btn' iconCss="e-
btn-icon e-icons e-delete-icon" onClick={onClick}></ButtonComponent>);
      }
    })()}
    </div>);
}

```

TEMPLATE.TSX

```

import * as React from 'react';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { getComponent, closest } from '@syncfusion/ej2-base';
import { CheckBoxComponent, ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DropDownButtonComponent } from '@syncfusion/ej2-react-splitbuttons';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import { QueryBuilder, ActionEventArgs, RuleModel } from '@syncfusion/ej2-querybuilder';
import { ItemModel, MenuEventArgs } from '@syncfusion/ej2-splitbuttons';
export function HeaderTemplate(props) {
    let ds: { [key: string]: Object }[] = [{ 'key': 'AND', 'value': 'and' }, { 'key': 'OR', 'value': 'or' }];
    let qryBldrObj: QueryBuilderComponent;
    let fields: object = { text: 'key', value: 'value' };
    let ddbItems: ItemModel[] = [
        {
            text: 'AddGroup',
            iconCss: 'e-icons e-add-icon e-addgroup'
        },
        {
            text: 'AddCondition',
            iconCss: 'e-icons e-add-icon e-addrule'
        }
    ];
    let state = Object.assign({}, props);
    qryBldrObj = getComponent(document.getElementById('querybuilder'), 'query-builder') as QueryBuilder;
    function onChange(args: any): void {
        qryBldrObj.notifyChange(args.checked, args.event.target, 'not');
    }
    function conditionChange(args: any): void {
        qryBldrObj.notifyChange(args.value, args.element, 'condition');
    }
    function onSelect(event: MenuEventArgs): void {
        let addbtn: Element = closest(event.element, '.e-dropdown-popup');
        let ddbId: string = addbtn.id; let ddb: string[] = ddbId.split('_');
        if (event.item.text === 'AddGroup') {
            qryBldrObj.addGroups([condition: 'or', rules: [{}], not: false], ddb[1]);
        } else if (event.item.text === 'AddCondition') {
            qryBldrObj.addRules([{}], ddb[1]);
        }
    }
    function onClick(args: any): void {
        qryBldrObj.deleteGroup(closest(args.target.offsetParent, '.e-group-container'));
    }
    const args: ActionEventArgs = state;
    return (<div className = "e-groupheader">
        {(() => {
            if (args.notCondition !== undefined) {
                return (<button className = "e-cb-wrapper">

```



```

        <CheckBoxComponent id = {args.ruleID + "_notOption"}
        label="not" checked={args.notCondition} change={onChange}/>
        </button>;
    }
    }) () }
    <DropDownListComponent id = {args.ruleID + "_cndtn"} cssClass = 'e-
    custom-group-btn' dataSource={ds} fields={fields} value={args.condition}
    change={conditionChange}/>
    <DropDownButtonComponent id = {args.ruleID + "_addbtn"}
    items={ddbitems} cssClass= "e-round e-small e-caret-hide e-addrulegroup e-
    add-btn" iconCss="e-icons e-add-icon"
    select={onSelect}></DropDownButtonComponent>
    {(()=> {
        if (args.ruleID !== "querybuilder_group0") {
            return(<ButtonComponent id = {args.ruleID + "_dltbtn"}
            cssClass = 'e-btn e-delete-btn e-lib e-small e-round e-icon-btn' iconCss="e-
            btn-icon e-icons e-delete-icon" onClick={onClick} ></ButtonComponent>)
        }
    }) () }
    </div>
);
}

```

Column Template

Template allows you to define your own input widgets for columns. To implement [template](#), you can define the following functions

- **create**: Creates the custom component.
- **write**: Wire events for the custom component.
- **Destroy**: Destroy the custom component.

In the following sample, dropdown is used as the custom component in the PaymentMode column.

APP.JSX

```

{% raw %}
import { getComponent } from '@syncfusion/ej2-base';
import { DropDownList, MultiSelect } from '@syncfusion/ej2-react-dropdowns';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { expenseData } from '../datasource.ts';
function App() {
    let qryBldrObj;
    let elem;
    let dropDownObj;
    let multiSelectObj;
    let inOperators = ['in', 'notin'];
    let filter = [
        {
            field: 'PaymentMode', label: 'Payment Mode', operators: [
                { key: 'Equal', value: 'equal' },
                { key: 'Not Equal', value: 'notequal' },
            ]
        }
    ]
}

```

```

        { key: 'In', value: 'in' },
        { key: 'Not In', value: 'notin' }
    ], template: {
        create: () => {
            elem = document.createElement('input');
            elem.setAttribute('type', 'text');
            return elem;
        },
        destroy: (args) => {
            multiSelectObj =
getComponent(document.getElementById(args.elementId), 'multiselect');
            if (multiSelectObj) {
                multiSelectObj.destroy();
            }
            dropDownObj =
getComponent(document.getElementById(args.elementId), 'dropdownlist');
            if (dropDownObj) {
                dropDownObj.destroy();
            }
        },
        write: (args) => {
            const ds = ['Cash', 'Debit Card', 'Credit Card', 'Net
Banking', 'Wallet'];
            if (inOperators.indexOf(args.operator) > -1) {
                multiSelectObj = new MultiSelect({
                    change: (e) => {
                        qryBldrObj.notifyChange(e.value, e.element);
                    },
                    dataSource: ds,
                    mode: 'CheckBox',
                    placeholder: 'Select Transaction',
                    value: args.values
                });
                multiSelectObj.appendTo('#' + args.elements.id);
            }
            else {
                dropDownObj = new DropDownList({
                    change: (e) => {
                        qryBldrObj.notifyChange(e.itemData.value,
e.element);
                    },
                    dataSource: ds,
                    value: args.values ? args.values : ds[0]
                });
                dropDownObj.appendTo('#' + args.elements.id);
            }
        }, type: 'string'
    },
    { field: 'Description', label: 'Description', type: 'string' },
    { field: 'Date', label: 'Date', type: 'date' }
];
let importRules = {
    'condition': 'or',
    'rules': [{
        'field': 'PaymentMode',
        'label': 'PaymentMode',

```

```

        'operator': 'equal',
        'type': 'string',
        'value': 'Cash'
    }
  ]
};
return (<QueryBuilderComponent dataSource={expenseData} columns={filter}
width='100%' rule={importRules} ref={({scope}) => { qryBldrObj = scope; }}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% endraw %}

```

APP.TSX

```

{% raw %}
import { getComponent } from '@syncfusion/ej2-base';
import { ChangeEventArgs, DropDownList, MultiSelect,
MultiSelectChangeEventArgs } from '@syncfusion/ej2-react-dropdowns';
import { ColumnsModel, QueryBuilderComponent, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { expenseData } from '../datasource.ts';
function App() {
  let qryBldrObj: QueryBuilderComponent;
  let elem: HTMLElement;
  let dropDownObj: DropDownList;
  let multiSelectObj: MultiSelect;
  let inOperators: string[] = ['in', 'notin'];
  let filter: ColumnsModel[] = [
    {
      field: 'PaymentMode', label: 'Payment Mode', operators: [
        { key: 'Equal', value: 'equal' },
        { key: 'Not Equal', value: 'notequal' },
        { key: 'In', value: 'in' },
        { key: 'Not In', value: 'notin' }
      ], template: {
        create: () => {
          elem = document.createElement('input');
          elem.setAttribute('type', 'text');
          return elem;
        },
        destroy: (args: { elementId: string }) => {
          multiSelectObj =
getComponent(document.getElementById(args.elementId) as HTMLElement,
'multiselect') as MultiSelect;
          if (multiSelectObj) {
            multiSelectObj.destroy();
          }
          dropDownObj =
getComponent(document.getElementById(args.elementId) as HTMLElement,
'dropdownlist') as DropDownList;
          if (dropDownObj) {
            dropDownObj.destroy();
          }
        }
      }
    }
  ];
}

```

```

        },
        write: (args: { elements: Element, values: string[] |
string, operator: string }) => {
            const ds = ['Cash', 'Debit Card', 'Credit Card', 'Net
Banking', 'Wallet'];
            if (inOperators.indexOf(args.operator) > -1) {
                multiSelectObj = new MultiSelect({
                    change: (e: MultiSelectChangeEventArgs) => {
                        qryBldrObj.notifyChange(e.value as string[],
e.element);
                    },
                    dataSource: ds,
                    mode: 'CheckBox',
                    placeholder: 'Select Transaction',
                    value: args.values as string []
                });
                multiSelectObj.appendTo('#' + args.elements.id);
            } else {
                dropDownObj = new DropDownList({
                    change: (e: ChangeEventArgs) => {
                        qryBldrObj.notifyChange(e.itemData.value as
string, e.element);
                    },
                    dataSource: ds,
                    value: args.values ? args.values as string :
ds[0]
                });
                dropDownObj.appendTo('#' + args.elements.id);
            }
        }, type: 'string'
    },
    { field: 'Description', label: 'Description', type: 'string' },
    { field: 'Date', label: 'Date', type: 'date' }
];
let importRules: RuleModel = {
    'condition': 'or',
    'rules': [{
        'field': 'PaymentMode',
        'label': 'PaymentMode',
        'operator': 'equal',
        'type': 'string',
        'value': 'Cash'
    }]
};
return (
    <QueryBuilderComponent dataSource={expenseData} columns={filter}
width='100%' rule={importRules} ref={(scope) => { qryBldrObj = scope as
QueryBuilderComponent; }} />
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% endraw %}

```

Using Template

Template allows you to define your own input widgets for columns. To implement template, you can create the user interface as `React` component.

APP.JSX

```
import { QueryBuilderComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PaymentTemplate } from './payment-temp';
import { TransactionTemplate } from './transaction-temp';
function App() {
  let qryBldrObj;
  let importRules = {
    'condition': 'and',
    'rules': [{
      'label': 'Transaction Type',
      'field': 'TransactionType',
      'type': 'string',
      'operator': 'equal',
      'value': 'Expense'
    },
    {
      'label': 'Payment Mode',
      'field': 'PaymentMode',
      'type': 'string',
      'operator': 'equal',
      'value': 'Cash'
    }
  ]
};
  function paymentTemplate(props) {
    return (<PaymentTemplate {...props}/>);
  }
  function transactionTemplate(props) {
    return (<TransactionTemplate {...props}/>);
  }
  return (<div>
    <QueryBuilderComponent width='100%' rule={importRules}
    id='querybuilder'>
      <ColumnsDirective>
        <ColumnDirective field="Category" label="Category"
        type="string"/>
        <ColumnDirective field="PaymentMode" label="PaymentMode"
        type="string" template={paymentTemplate}/>
        <ColumnDirective field="TransactionType"
        label="TransactionType" type="string" template={transactionTemplate}/>
        <ColumnDirective field="Description" label="Description"
        type="string"/>
        <ColumnDirective field="Date" label="Date" type="string"/>
        <ColumnDirective field="Amount" label="Amount"
        type="string"/>
      </ColumnsDirective>
    </QueryBuilderComponent>
  </div>);
}
export default App;
```

```
ReactDOM.render(<App />, document.getElementById('querybuilder-component'));
```

APP.TSX

```
import { QueryBuilderComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-react-querybuilder';
import { RuleModel } from '@syncfusion/ej2-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { PaymentTemplate } from './payment-temp';
import { TransactionTemplate } from './transaction-temp';
function App() {
  let qryBldrObj: QueryBuilderComponent;
  let importRules: RuleModel = {
    'condition': 'and',
    'rules': [{
      'label': 'Transaction Type',
      'field': 'TransactionType',
      'type': 'string',
      'operator': 'equal',
      'value': 'Expense'
    },
    {
      'label': 'Payment Mode',
      'field': 'PaymentMode',
      'type': 'string',
      'operator': 'equal',
      'value': 'Cash'
    }
  ]
};
function paymentTemplate(props) {
  return (<PaymentTemplate {...props}/>);
}
function transactionTemplate(props) {
  return (<TransactionTemplate {...props}/>);
}
return (<div>
  <QueryBuilderComponent width='100%' rule={importRules}
id='querybuilder' >
    <ColumnsDirective>
      <ColumnDirective field="Category" label="Category"
type="string"/>
      <ColumnDirective field="PaymentMode" label="PaymentMode"
type="string" template = {paymentTemplate}/>
      <ColumnDirective field="TransactionType"
label="TransactionType" type="string" template = {transactionTemplate}/>
      <ColumnDirective field="Description" label="Description"
type="string"/>
      <ColumnDirective field="Date" label="Date" type="string"/>
      <ColumnDirective field="Amount" label="Amount"
type="string"/>
    </ColumnsDirective>
  </QueryBuilderComponent>
</div>);
}
export default App;
```

```
ReactDOM.render(<App />, document.getElementById('querybuilder-component'));
```

PAYMENT-TEMP.JSX

```
import * as React from 'react';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { getComponent } from '@syncfusion/ej2-base';
export function PaymentTemplate(props) {
    let ds = ['Cash', 'Debit Card', 'Credit Card', 'Net Banking'];
    let qryBldrObj;
    let state = Object.assign({}, props);
    qryBldrObj = getComponent(document.getElementById('querybuilder'),
    'query-builder');
    const args = state;
    function paymentChange(event) {
        const args = state;
        let elem = document.getElementById(args.ruleID).querySelector('.e-
rule-value');
        qryBldrObj.notifyChange(event.value, elem, 'value');
    }
    return (<div>
        <DropDownListComponent dataSource={ds} value={args.rule.value}
change={paymentChange}/>
        </div>);
}
```

PAYMENT-TEMP.TSX

```
import * as React from 'react';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { getComponent, closest } from '@syncfusion/ej2-base';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import { QueryBuilder, ActionEventArgs, RuleModel } from '@syncfusion/ej2-
querybuilder';
export function PaymentTemplate(props) {
    let ds: string[] = ['Cash', 'Debit Card', 'Credit Card', 'Net Banking'];
    let qryBldrObj: QueryBuilderComponent;
    let state = Object.assign({}, props);
    qryBldrObj = getComponent(document.getElementById('querybuilder'),
    'query-builder') as QueryBuilder;
    const args: ActionEventArgs = state;
    function paymentChange(event: any): void{
        const args: ActionEventArgs = state;
        let elem: HTMLElement =
document.getElementById(args.ruleID).querySelector('.e-rule-value');
        qryBldrObj.notifyChange(event.value as string, elem, 'value');
    }
    return (<div >
        <DropDownListComponent dataSource={ds} value={args.rule.value}
change={paymentChange}/>
        </div>
    );
}
```

TRANSACTION-TEMP.JSX

```
import * as React from 'react';
import { getComponent } from '@syncfusion/ej2-base';
import { CheckBoxComponent } from '@syncfusion/ej2-react-buttons';
export function TransactionTemplate(props) {
    let qryBldrObj;
    let state = Object.assign({}, props);
    qryBldrObj = getComponent(document.getElementById('querybuilder'),
    'query-builder');
    const args = state;
    function transactionChange(event) {
        const args = state;
        let elem = document.getElementById(args.ruleID).querySelector('.e-
rule-value');
        qryBldrObj.notifyChange(event.checked === true ? 'Expense' :
'Income', elem, 'value');
    }
    return (<div>
        <CheckBoxComponent label="Is Expense" checked={args.rule.value}
change={transactionChange}/>
    </div>);
}
```

TRANSACTION-TEMP.TSX

```
import * as React from 'react';
import { getComponent, closest } from '@syncfusion/ej2-base';
import { CheckBoxComponent, ButtonComponent } from '@syncfusion/ej2-react-
buttons';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import { QueryBuilder, ActionEventArgs, RuleModel } from '@syncfusion/ej2-
querybuilder';
export function TransactionTemplate(props) {
    let qryBldrObj: QueryBuilderComponent;
    let state = Object.assign({}, props);
    qryBldrObj = getComponent(document.getElementById('querybuilder'),
    'query-builder') as QueryBuilder;
    const args: ActionEventArgs = state;
    function transactionChange(event: any): void {
        const args: ActionEventArgs = state;
        let elem: HTMLElement =
document.getElementById(args.ruleID).querySelector('.e-rule-value');
        qryBldrObj.notifyChange(event.checked === true ? 'Expense' :
'Income', elem, 'value');
    }
    return (<div>
        <CheckBoxComponent label="Is Expense" checked={args.rule.value}
change={transactionChange}/>
    </div>
    );
}
```

Rule Template

Rule Template allows to define your own user interface for columns. To implement [ruleTemplate](#), you can create the user interface as **React** component and assign the values through **actionBegin** event.

In the following sample, dropdown and slider are used as the custom component and applied `greaterthanorequal` operator to `Age` column.

APP.JSX

```
{% raw %}
import { QueryBuilderComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
import { AgeTemplate } from './template';
function App() {
  let qryBldrObj;
  let elem;
  let importRules = {
    'condition': 'or',
    'rules': [{
      'field': 'Age',
      'label': 'Age',
      'operator': 'greaterthanorequal',
      'type': 'number',
      'value': 30
    }]
  };
  function ageTemplate(props) {
    return (<AgeTemplate {...props}/>);
  }
  function actionBegin(args) {
    let ruleID = args.ruleID;
    args.rule.operator = 'greaterthanorequal';
    if (args.requestType === 'template-initialize') {
      if (args.rule.value === '') {
        args.rule.value = 30;
      }
    }
  }
  return (<QueryBuilderComponent dataSource={employeeData} width='100%'
    rule={importRules} ref={(scope) => { qryBldrObj = scope; }}
    actionBegin={actionBegin} id='querybuilder'>
    <ColumnsDirective>
      <ColumnDirective field="EmployeeID" label="Employee ID"
type="number"/>
      <ColumnDirective field="LastName" label="Last Name"
type="string"/>
      <ColumnDirective field="FirstName" label="First Name"
type="string"/>
      <ColumnDirective field="Age" label="Age" type="number"
ruleTemplate={ageTemplate}/>
      <ColumnDirective field="City" label="City" type="string"/>
      <ColumnDirective field="Country" label="Country"
type="string"/>
    </ColumnsDirective>
  </QueryBuilderComponent>);
}
export default App;
```

```
ReactDOM.render(<App />, document.getElementById('querybuilder-component'));
{% endraw %}
```

APP.TSX

```
{% raw %}
import { getComponent, setValue } from '@syncfusion/ej2-base';
import { ChangeEventArgs, DropDownList } from '@syncfusion/ej2-react-
dropdowns';
import { QueryBuilderComponent, ColumnsDirective, ColumnDirective,
RuleModel, ActionEventArgs } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
import { AgeTemplate } from './template';
function App() {
    let qryBldrObj: QueryBuilderComponent;
    let elem: HTMLElement;
    let importRules: RuleModel = {
        'condition': 'or',
        'rules': [{
            'field': 'Age',
            'label': 'Age',
            'operator': 'greaterthanorequal',
            'type': 'number',
            'value': 30
        }]
    };
    function ageTemplate(props): any {
        return (<AgeTemplate {...props} />);
    }
    function actionBegin(args: ActionEventArgs): void {
        let ruleID = args.ruleID;
        args.rule.operator = 'greaterthanorequal';
        if (args.requestType === 'template-initialize') {
            if (args.rule.value === '') {
                args.rule.value = 30;
            }
        }
    }
    return (
        <QueryBuilderComponent dataSource={employeeData} width='100%'
        rule={importRules} ref={(scope) => { qryBldrObj = scope as
QueryBuilderComponent; }} actionBegin={actionBegin} id='querybuilder'>
            <ColumnsDirective>
                <ColumnDirective field="EmployeeID" label="Employee ID"
type="number"/>
                <ColumnDirective field="LastName" label="Last Name"
type="string"/>
                <ColumnDirective field="FirstName" label="First Name"
type="string"/>
                <ColumnDirective field="Age" label="Age" type="number"
ruleTemplate={ageTemplate}/>
                <ColumnDirective field="City" label="City" type="string"/>
            </ColumnsDirective>
        </QueryBuilderComponent>
    );
}
```

```

        <ColumnDirective field="Country" label="Country"
type="string"/>
        </ColumnsDirective>
    </QueryBuilderComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder-component'));
{% endraw %}

```

TEMPLATE.JSX

```

{% raw %}
import * as React from 'react';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { getComponent, compile } from '@syncfusion/ej2-base';
import { DataManager, Query } from '@syncfusion/ej2-data';
import { employeeData } from '../datasource.ts';
export function AgeTemplate(props) {
    let qryBldrObj;
    let sliderObj;
    let rangeTicks = { placement: 'Before', largeStep: 5, showSmallTicks:
true };
    let state = Object.assign({}, props);
    qryBldrObj = getComponent(document.getElementById('querybuilder'),
'query-builder');
    function fieldChange(args) {
        qryBldrObj.notifyChange(args.value, args.element, 'field');
    }
    function valueChange(args) {
        let elem = sliderObj.element;
        qryBldrObj.notifyChange(args.value, elem, 'value');
        refreshTable(qryBldrObj.getRule(elem),
elem.id.split('_valuekey0')[0]);
    }
    function sliderCreated() {
        let elem = sliderObj.element;
        refreshTable(qryBldrObj.getRule(elem),
elem.id.split('_valuekey0')[0]);
    }
    function myFunction(ruleID) {
        let element = document.getElementById(ruleID + '_section');
        if (element.className.indexOf('e-hide') > -1) {
            element.className = element.className.replace('e-hide', '');
            document.getElementById(ruleID + '_option').textContent = 'Hide
Details';
        }
        else {
            element.className += ' e-hide';
            document.getElementById(ruleID + '_option').textContent = 'View
Details';
        }
    }
    function refreshTable(rule, ruleID) {

```

```

    let template =
    '<tr><td>${EmployeeID}</td><td>${FirstName}</td><td>${Age}</td></tr>';
    let compiledFunction = compile(template);
    let predicate = qryBldrObj.getPredicate({ condition: 'and', rules:
[rule] });
    let dataManagerQuery = new Query().select(['EmployeeID',
'FirstName', 'Age']).where(predicate);
    let result = new
DataManager(employeeData).executeLocal(dataManagerQuery);
    let table = document.getElementById(ruleID + '_datatable');
    if (result.length) {
        table.style.display = 'block';
    }
    else {
        table.style.display = 'none';
    }
    table.querySelector('tbody').innerHTML = '';
    result.forEach((data) => {

table.querySelector('tbody').appendChild(compiledFunction(data)[0].querySele
ctor('tr'));
    });
    }
    const args = state;
    return (<div className="e-rule e-rule-template">
        <div className="e-rule-filter e-custom-filter">
            <DropDownListComponent change={fieldChange} fields={args.fields}
dataSource={args.columns} value={args.rule.field}/>
        </div>
        <div>
            <div className="e-slider-value">
                <SliderComponent ticks={rangeTicks} ref={(scope) => { sliderObj =
scope; }} id={args.ruleID + '_valuekey0'} change={valueChange}
created={sliderCreated} value={args.rule.value} min={30} max={50}/>
            </div>
            <div className="e-rule-btn">
                <button id={args.ruleID + '_option'} onClick={myFunction.bind(this,
args.ruleID)} className="e-primary e-btn e-small">
                    View Details
                </button>
                <button className="e-remove-rule e-rule-delete e-css e-btn e-small e-
round">
                    <span className="e-btn-icon e-icons e-delete-icon"/>
                </button>
            </div>
        </div>
        <div id={args.ruleID + '_section'} className="e-rule-value-group
e-hide">
            <div>
                <table id={args.ruleID + '_datatable'} className='e-rule-
table e-hide'>
                    <thead>
<tr><th>EmployeeID</th><th>FirstName</th><th>Age</th></tr>
                    </thead>
                    <tbody>
</tbody>
                </table>
            </div>

```

```

        </table>
      </div>
    </div>
  </div>);
}
{% endraw %}

```

TEMPLATE.TSX

```

{% raw %}
import * as React from 'react';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { getComponent, compile } from '@syncfusion/ej2-base';
import { DataManager, Predicate, Query } from '@syncfusion/ej2-data';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import { QueryBuilder, ActionEventArgs, RuleModel } from '@syncfusion/ej2-querybuilder';
import { employeeData } from '../datasource.ts';
export function AgeTemplate(props) {
  let qryBldrObj: QueryBuilderComponent;
  let sliderObj: SliderComponent;
  let rangeTicks: object = { placement: 'Before', largeStep: 5,
showSmallTicks: true };
  let state = Object.assign({}, props);
  qryBldrObj = getComponent(document.getElementById('querybuilder'), 'query-builder') as QueryBuilder;
  function fieldChange(args: any): void {
    qryBldrObj.notifyChange(args.value, args.element, 'field');
  }
  function valueChange(args: any): void {
    let elem: Element = sliderObj.element;
    qryBldrObj.notifyChange(args.value, elem, 'value');
    refreshTable(qryBldrObj.getRule(elem),
elem.id.split('_valuekey0')[0]);
  }
  function sliderCreated() {
    let elem: Element = sliderObj.element;
    refreshTable(qryBldrObj.getRule(elem),
elem.id.split('_valuekey0')[0]);
  }
  function myFunction(ruleID: string): void {
    let element: Element = document.getElementById(ruleID + '_section');
    if (element.className.indexOf('e-hide') > -1) {
      element.className = element.className.replace('e-hide', '');
      document.getElementById(ruleID + '_option').textContent = 'Hide
Details';
    } else {
      element.className += ' e-hide';
      document.getElementById(ruleID + '_option').textContent = 'View
Details';
    }
  }
  function refreshTable(rule: RuleModel, ruleID: string): void {
    let template: string =
'<tr><td>${EmployeeID}</td><td>${FirstName}</td><td>${Age}</td></tr>';

```

```

        let compiledFunction: any = compile(template);
        let predicate: Predicate = gryBldrObj.getPredicate({condition: 'and',
rules: [rule]});
        let dataManagerQuery: Query = new Query().select(['EmployeeID',
'FirstName', 'Age']).where(predicate);
        let result: object[] = new
DataManager(employeeData).executeLocal(dataManagerQuery);
        let table: HTMLElement = document.getElementById(ruleID +
'_datatable') as HTMLElement;
        if (result.length) {
            table.style.display = 'block';
        } else {
            table.style.display = 'none';
        }
        table.querySelector('tbody').innerHTML = '';
        result.forEach((data) => {

table.querySelector('tbody').appendChild(compiledFunction(data) [0].querySele
ctor('tr'));
        });
    }
    const args: ActionEventArgs = state;
    return (
        <div className="e-rule e-rule-template">
            <div className="e-rule-filter e-custom-filter">
                <DropDownListComponent change={fieldChange} fields={args.fields}
dataSource={args.columns} value={args.rule.field}/>
            </div>
            <div>
                <div className="e-slider-value">
                    <SliderComponent ticks={rangeTicks} ref={(scope) => { sliderObj =
scope as SliderComponent; }} id={args.ruleID + '_valuekey0'}
change={valueChange} created={sliderCreated} value={args.rule.value}
min={30} max={50} />
                </div>
                <div className="e-rule-btn">
                    <button id={args.ruleID + '_option'} onClick={myFunction.bind(this,
args.ruleID)} className="e-primary e-btn e-small">
                        View Details
                    </button>
                    <button className="e-removerule e-rule-delete e-css e-btn e-small e-
round">
                        <span className="e-btn-icon e-icons e-delete-icon"/>
                    </button>
                </div>
            </div>
            <div id={args.ruleID + '_section'} className="e-rule-value-group
e-hide">
                <div>
                    <table id={args.ruleID + '_datatable'} className='e-rule-
table e-hide'>
                        <thead>
<tr><th>EmployeeID</th><th>FirstName</th><th>Age</th></tr>
                        </thead>
                        <tbody>
</tbody>
                    </table>
                </div>
            </div>
        </div>
    );

```

```

        </table>
      </div>
    </div>
  </div>
);
}{{% enddraw %}}

```

Model binding in React Query builder component

Model binding allows to bind properties for the components used in field, operator, and value columns. To implement model binding, assign fieldModel, operatorModel, and valueModel properties in QueryBuilder.

APP.JSX

```

import { QueryBuilderComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let field = {
    allowFiltering: true,
    popupHeight: '400px'
  };
  let operator = {
    allowFiltering: true,
    popupHeight: '400px'
  };
  let value = {
    numericTextBoxModel: {
      cssClass: 'e-custom'
    },
    multiSelectModel: {
      cssClass: 'e-custom'
    },
    datePickerModel: {
      cssClass: 'e-custom'
    },
    textBoxModel: {
      cssClass: 'e-custom'
    },
    radioButtonModel: {
      cssClass: 'e-custom'
    }
  };
  let importRules = {
    'condition': 'and',
    'rules': [{
      'label': 'Employee ID',
      'field': 'EmployeeID',
      'type': 'number',
      'operator': 'equal',
      'value': 1001
    }]
  };
  return <div>

```

```

    <QueryBuilderComponent width='100%' rule={importRules}
    id='querybuilder' enableNotCondition="true" fieldModel={field}
    operatorModel={operator} valueModel={value}>
      <ColumnsDirective>
        <ColumnDirective field="EmployeeID" label="Employee ID"
type="number"/>
        <ColumnDirective field="LastName" label="Last Name"
type="string"/>
        <ColumnDirective field="FirstName" label="First Name"
type="string"/>
        <ColumnDirective field="Age" label="Age" type="number"/>
        <ColumnDirective field="City" label="City" type="string"/>
        <ColumnDirective field="Country" label="Country"
type="string"/>
      </ColumnsDirective>
    </QueryBuilderComponent>
  </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder-component'));

```

APP.TSX

```

import { QueryBuilderComponent, ColumnsDirective, ColumnDirective } from
'@syncfusion/ej2-react-querybuilder';
import { RuleModel, ColumnsModel, ValueModel } from '@syncfusion/ej2-
querybuilder';
import { DropDownListModel } from '@syncfusion/ej2-dropdowns';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  let field: DropDownListModel = {
    allowFiltering : true,
    popupHeight: '400px'
  }
  let operator: DropDownListModel= {
    allowFiltering : true,
    popupHeight: '400px'
  }
  let value: ValueModel = {
    numericTextBoxModel:{
      cssClass: 'e-custom'
    },
    multiSelectModel: {
      cssClass: 'e-custom'
    },
    datePickerModel: {
      cssClass: 'e-custom'
    },
    textBoxModel: {
      cssClass: 'e-custom'
    },
    radioButtonModel: {
      cssClass: 'e-custom'
    }
  }
}

```



```

let importRules: RuleModel = {
  'condition': 'and',
  'rules': [{
    'label': 'Employee ID',
    'field': 'EmployeeID',
    'type': 'number',
    'operator': 'equal',
    'value': 1001
  }]
};
return (<div>
  <QueryBuilderComponent width='100%' rule={importRules}
id='querybuilder' enableNotCondition ="true" fieldModel ={field}
operatorModel = {operator} valueModel = {value} >
    <ColumnsDirective>
      <ColumnDirective field="EmployeeID" label="Employee ID"
type="number"/>
      <ColumnDirective field="LastName" label="Last Name"
type="string"/>
      <ColumnDirective field="FirstName" label="First Name"
type="string"/>
      <ColumnDirective field="Age" label="Age" type="number"/>
      <ColumnDirective field="City" label="City" type="string"/>
      <ColumnDirective field="Country" label="Country"
type="string"/>
    </ColumnsDirective>
  </QueryBuilderComponent>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder-component'));

```

Importing and Exporting in React Query builder component

Importing facilitates the viewing or editing of predefined conditions available in JSON, SQL, and MongoDB query formats, while exporting enables obtaining the created rules in the query builder as JSON, SQL, and MongoDB queries.

Importing

Importing enables users to bring predefined conditions into the system for viewing or editing, available in formats such as JSON, SQL, and MongoDB query. It facilitates the quick incorporation of pre-defined rules or parameters into workflows, streamlining the setup process by importing directly from external sources or saved configurations.

Importing from JSON Object

Importing from JSON enables users to bring predefined conditions encoded in JSON format into the system. This feature streamlines the process by providing a standardized format for importing data, ensuring compatibility, and ease of use.

Initial rendering

To initially apply conditions, you can establish the [rule](#) by importing a structured JSON object and defining its properties.

APP.JSX

```
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
```

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
  let columnData = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  let importRules = {
    'condition': 'or',
    'rules': [{
      'field': 'Category',
      'label': 'Category',
      'operator': 'equal',
      'type': 'string',
      'value': 'Laptop'
    },
    {
      'condition': 'and',
      'rules': [{
        'field': 'Status',
        'label': 'Status',
        'operator': 'notequal',
        'type': 'string',
        'value': 'Pending'
      },
      {
        'field': 'TaskID',
        'label': 'Task ID',
        'operator': 'equal',
        'type': 'number',
        'value': 5675
      }
    ]
  ]
  };
  return (<QueryBuilderComponent width='100%' dataSource={hardwareData}
  columns={columnData} rule={importRules}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
  let columnData: ColumnsModel[] = [

```

```

        { field: 'TaskID', label: 'Task ID', type: 'number' },
        { field: 'Name', label: 'Name', type: 'string' },
        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
    let importRules: RuleModel = {
        'condition': 'or',
        'rules': [{
            'field': 'Category',
            'label': 'Category',
            'operator': 'equal',
            'type': 'string',
            'value': 'Laptop'
        },
        {
            'condition': 'and',
            'rules': [{
                'field': 'Status',
                'label': 'Status',
                'operator': 'notequal',
                'type': 'string',
                'value': 'Pending'
            },
            {
                'field': 'TaskID',
                'label': 'Task ID',
                'operator': 'equal',
                'type': 'number',
                'value': 5675
            }
        ]
    }
    ];

    return (
        <QueryBuilderComponent width='100%' dataSource={hardwareData}
        columns={columnData} rule={importRules}/>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Post rendering

You can set the conditions from structured JSON object through the [setRules](#) method.

APP.JSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {

```

```

let gryBldrObj;
let columnData = [
  { field: 'TaskID', label: 'Task ID', type: 'number' },
  { field: 'Name', label: 'Name', type: 'string' },
  { field: 'Category', label: 'Category', type: 'string' },
  { field: 'SerialNo', label: 'Serial No', type: 'string' },
  { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
  { field: 'Status', label: 'Status', type: 'string' }
];
let importRules = {
  'condition': 'or',
  'rules': [{
    'field': 'Category',
    'label': 'Category',
    'operator': 'equal',
    'type': 'string',
    'value': 'Laptop'
  },
  {
    'condition': 'and',
    'rules': [{
      'field': 'Status',
      'label': 'Status',
      'operator': 'notequal',
      'type': 'string',
      'value': 'Pending'
    },
    {
      'field': 'TaskID',
      'label': 'Task ID',
      'operator': 'equal',
      'type': 'number',
      'value': 5675
    }
  ]
}
];
function onClick() {
  gryBldrObj.setRules(importRules);
}
return (<div>
  <QueryBuilderComponent width='100%' dataSource={hardwareData}
  ref={(scope) => { gryBldrObj = scope; }} columns={columnData}/>
  <div className="e-qb-button">
    <ButtonComponent id="importrules" cssClass='e-primary'
    content='Set Rules' onClick={onClick}/>
  </div>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';

```

```

import { ColumnsModel, QueryBuilderComponent, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
    let qryBldrObj: QueryBuilderComponent;
    let columnData: ColumnsModel[] = [
        { field: 'TaskID', label: 'Task ID', type: 'number' },
        { field: 'Name', label: 'Name', type: 'string' },
        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
    let importRules: RuleModel = {
        'condition': 'or',
        'rules': [{
            'field': 'Category',
            'label': 'Category',
            'operator': 'equal',
            'type': 'string',
            'value': 'Laptop'
        },
        {
            'condition': 'and',
            'rules': [{
                'field': 'Status',
                'label': 'Status',
                'operator': 'notequal',
                'type': 'string',
                'value': 'Pending'
            },
            {
                'field': 'TaskID',
                'label': 'Task ID',
                'operator': 'equal',
                'type': 'number',
                'value': 5675
            }
        ]
    }
    ];
    function onClick(): void {
        qryBldrObj.setRules(importRules);
    }
    return (
        <div>
            <QueryBuilderComponent width='100%' dataSource={hardwareData}
            ref={(scope) => { qryBldrObj = scope as QueryBuilderComponent; }}
            columns={columnData}/>
            <div className="e-qb-button">
                <ButtonComponent id="importrules" cssClass='e-primary'
                content='Set Rules' onClick = {onClick}/>
            </div>
        </div>
    );
}

```

```

}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% endraw %}

```

Importing from SQL Query

Importing from SQL involves integrating predefined conditions or data stored in a SQL database into the Query Builder. This enables the direct integration of SQL queries, thereby improving workflow efficiency and data accuracy within the application. SQL importing supports various types, including Inline SQL, Parameter SQL, and Named Parameter SQL.

Importing from Inline SQL Query

Importing from Inline SQL involves integrating SQL queries directly into the Query Builder. This method streamlines the process by enabling users to input SQL statements directly into the application for analysis, manipulation, or further processing within the Query Builder. Conditions can be set from Inline SQL queries using the [setRulesFromSql](#) method.

APP.JSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
  let qryBldrObj;
  let columnData = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  function importRule() {
    qryBldrObj.setRulesFromSql("TaskID = 1 and Status LIKE ('Assigned%')");
  }
  return (<div>
    <QueryBuilderComponent width='100%' dataSource={hardwareData}
    ref={(scope) => { qryBldrObj = scope; }} columns={columnData}/>
    <div className="e-qb-button">
      <ButtonComponent id="importrules" cssClass='e-primary'
      content='set Rules' onClick={importRule}/>
    </div>
  </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% endraw %}

```

APP.TSX

```
{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { QueryBuilderComponent, ColumnsModel } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
    let qryBldrObj: QueryBuilderComponent;
    let columnData: ColumnsModel[] = [
        { field: 'TaskID', label: 'Task ID', type: 'number' },
        { field: 'Name', label: 'Name', type: 'string' },
        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
    function importRule(): void {
        qryBldrObj.setRulesFromSql("TaskID = 1 and Status LIKE ('Assigned%')");
    }
    return (
        <div>
            <QueryBuilderComponent width='100%' dataSource={hardwareData}
ref={(scope) => { qryBldrObj = scope as QueryBuilderComponent; }}
columns={columnData}/>
            <div className="e-qb-button">
                <ButtonComponent id="importrules" cssClass='e-primary'
content='set Rules' onClick = {importRule}/>
            </div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% endraw %}
```

Importing from Parameter SQL Query

Importing from Parameter SQL involves integrating SQL queries with parameters directly into the Query Builder. This method allows users to input SQL statements containing parameters, which can be dynamically filled in during execution. It streamlines the process by enabling flexible and customizable querying within the application. Conditions can be set from Parameter SQL queries using the [setParameterizedSql](#) method.

APP.JSX

```
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
function App() {
    let qryBldrObj;
```

```

let columnData = [
  { field: 'TaskID', label: 'Task ID', type: 'number' },
  { field: 'Name', label: 'Name', type: 'string' },
  { field: 'Category', label: 'Category', type: 'string' },
  { field: 'SerialNo', label: 'Serial No', type: 'string' },
  { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
  { field: 'Status', label: 'Status', type: 'string' }
];
let hardwareData = [{
  'TaskID': 1,
  'Name': 'Lenovo Yoga',
  'Category': 'Laptop',
  'SerialNo': 'CB27932009',
  'InvoiceNo': 'INV-2878',
  'Status': 'Assigned'
},
{
  'TaskID': 2,
  'Name': 'Acer Aspire',
  'Category': 'Others',
  'SerialNo': 'CB35728290',
  'InvoiceNo': 'INV-3456',
  'Status': 'In-repair'
},
{
  'TaskID': 3,
  'Name': 'Apple MacBook',
  'Category': 'Laptop',
  'SerialNo': 'CB35628728',
  'InvoiceNo': 'INV-2763',
  'Status': 'In-repair'
}
];

function importSql() {
  qryBldrObj.setParameterizedSql({ sql: '(Category IN (?,?) OR TaskID IN (?,?))', params: ['Laptop', 'Others', 1, 2] });
}
return (<div>
  <QueryBuilderComponent width='100%' dataSource={hardwareData}
columns={columnData} ref={(scope) => { qryBldrObj = scope; }}/>
  <div className="e-qb-button">
    <ButtonComponent id="importSql" cssClass='e-primary'
content='Set Parameter SQL Rules' onClick = {importSql}/>
  </div>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, QueryLibrary } from
'@syncfusion/ej2-react-querybuilder';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```



```

QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
function App() {
  let qryBldrObj: QueryBuilderComponent;
  let hardwareData: Object[] = [{
    'TaskID': 1,
    'Name': 'Lenovo Yoga',
    'Category': 'Laptop',
    'SerialNo': 'CB27932009',
    'InvoiceNo': 'INV-2878',
    'Status': 'Assigned'
  },
  {
    'TaskID': 2,
    'Name': 'Acer Aspire',
    'Category': 'Others',
    'SerialNo': 'CB35728290',
    'InvoiceNo': 'INV-3456',
    'Status': 'In-repair'
  },
  {
    'TaskID': 3,
    'Name': 'Apple MacBook',
    'Category': 'Laptop',
    'SerialNo': 'CB35628728',
    'InvoiceNo': 'INV-2763',
    'Status': 'In-repair'
  }
  ];
  let columnData: ColumnsModel[] = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  function importSql(): void {
    qryBldrObj.setParameterizedSql({ sql: '(Category IN (?,?) OR TaskID IN (?,?))', params: ['Laptop', 'Others', 1, 2] });
  }
  return (
    <div>
      <QueryBuilderComponent width='100%' dataSource={hardwareData}
        columns={columnData} ref={(scope) => { qryBldrObj = scope as
        QueryBuilderComponent; }}/>
      <div className="e-qb-button">
        <ButtonComponent id="importSql" cssClass='e-primary'
        content='Set Parameter SQL Rules' onClick = {importSql}/>
      </div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Importing from Named Parameter SQL Query

Importing from Named Parameter SQL involves integrating SQL queries with named parameters directly into the Query Builder. This method enables users to input SQL statements containing named parameters, providing flexibility and customization during execution. It streamlines the process by allowing dynamic parameter assignment within the application's query environment. Conditions can be set from Named Parameter SQL queries using the [setParameterizedNamedSql](#) method.

APP.JSX

```
import { QueryBuilderComponent, QueryLibrary } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
function App() {
  let qryBldrObj;
  let columnData = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  let hardwareData = [{
    'TaskID': 1,
    'Name': 'Lenovo Yoga',
    'Category': 'Laptop',
    'SerialNo': 'CB27932009',
    'InvoiceNo': 'INV-2878',
    'Status': 'Assigned'
  },
  {
    'TaskID': 2,
    'Name': 'Acer Aspire',
    'Category': 'Others',
    'SerialNo': 'CB35728290',
    'InvoiceNo': 'INV-3456',
    'Status': 'In-repair'
  },
  {
    'TaskID': 3,
    'Name': 'Apple MacBook',
    'Category': 'Laptop',
    'SerialNo': 'CB35628728',
    'InvoiceNo': 'INV-2763',
    'Status': 'In-repair'
  }
  ];

  function importSql() {
    qryBldrObj.setParameterizedNamedSql({ sql: '(Category IN (:Category_1,:Category_2) OR TaskID IN (:TaskID_1,:TaskID_2))', params:
    {"Category_1": "Laptop", "Category_2": "Others", "TaskID_1": 1, "TaskID_2": 2} });
  }
}
```

```

    }
    return (<div>
      <QueryBuilderComponent width='100%' dataSource={hardwareData}
        columns={columnData} ref={(scope) => { qryBldrObj = scope; }}/>
      <div className="e-qb-button">
        <ButtonComponent id="importSql" cssClass='e-primary'
          content='Set Named Parameter SQL Rules' onClick = {importSql}/>
      </div>
    </div>);
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, QueryLibrary } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
function App() {
  let qryBldrObj: QueryBuilderComponent;
  let hardwareData: Object[] = [{
    'TaskID': 1,
    'Name': 'Lenovo Yoga',
    'Category': 'Laptop',
    'SerialNo': 'CB27932009',
    'InvoiceNo': 'INV-2878',
    'Status': 'Assigned'
  },
  {
    'TaskID': 2,
    'Name': 'Acer Aspire',
    'Category': 'Others',
    'SerialNo': 'CB35728290',
    'InvoiceNo': 'INV-3456',
    'Status': 'In-repair'
  },
  {
    'TaskID': 3,
    'Name': 'Apple MacBook',
    'Category': 'Laptop',
    'SerialNo': 'CB35628728',
    'InvoiceNo': 'INV-2763',
    'Status': 'In-repair'
  }
  ];
  let columnData: ColumnsModel[] = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
}

```

```

function importSql(): void {
    gryBldrObj.setParameterizedNamedSql({ sql: '(Category IN
(:Category_1,:Category_2) OR TaskID IN (:TaskID_1,:TaskID_2))', params:
{"Category_1": "Laptop", "Category_2": "Others", "TaskID_1": 1, "TaskID_2":
2} });
}
return (
    <div>
        <QueryBuilderComponent width='100%' dataSource={hardwareData}
columns={columnData} ref={(scope) => { gryBldrObj = scope as
QueryBuilderComponent; }}/>
        <div className="e-qb-button">
            <ButtonComponent id="clonegroup" cssClass='e-primary'
content='Set Named Parameter SQL Rules' onClick = {importSql}/>
        </div>
    </div>
);
}
export default App;
ReactDOM.render(<App />,document.getElementById('querybuilder'));

```

Importing from MongoDB Query

Importing from MongoDB Query involves integrating MongoDB queries directly into the Query Builder. This enables users to input MongoDB query statements directly into the application, allowing for seamless integration and manipulation of MongoDB data within the Query Builder environment. It streamlines the process by facilitating direct access to MongoDB data for analysis, filtering, and further processing within the application. Conditions can be set from Named Parameter SQL queries using the [setMongoQuery](#) method.

APP.JSX

```

import { QueryBuilderComponent, QueryLibrary } from '@syncfusion/ej2-react-
querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
    let gryBldrObj;
    let columnData = [
        { field: 'TaskID', label: 'Task ID', type: 'number' },
        { field: 'Name', label: 'Name', type: 'string' },
        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
    let hardwareData = [{
        'TaskID': 1,
        'Name': 'Lenovo Yoga',
        'Category': 'Laptop',
        'SerialNo': 'CB27932009',
        'InvoiceNo': 'INV-2878',
        'Status': 'Assigned'
    }];
}

```

```

    },
    {
      'TaskID': 2,
      'Name': 'Acer Aspire',
      'Category': 'Others',
      'SerialNo': 'CB35728290',
      'InvoiceNo': 'INV-3456',
      'Status': 'In-repair'
    },
    {
      'TaskID': 3,
      'Name': 'Apple MacBook',
      'Category': 'Laptop',
      'SerialNo': 'CB35628728',
      'InvoiceNo': 'INV-2763',
      'Status': 'In-repair'
    }
  ]];

  function importMongo() {
    gryBldrObj.setMongoQuery('{ "$and": [{"TaskID":1001},{
"$or":[{"Category":{"$regex":"Order"}}]}]}');
  }

  return (<div>
    <QueryBuilderComponent width='100%' dataSource={hardwareData}
columns={columnData} ref={ (scope) => { gryBldrObj = scope; } }/>
    <div className="e-qb-button">
      <ButtonComponent id="importSql" cssClass='e-primary'
content='Set MongoDB Rules' onClick = {importMongo}/>
    </div>
  </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, QueryLibrary } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
function App() {
  let gryBldrObj: QueryBuilderComponent;
  let hardwareData: Object[] = [{
    'TaskID': 1,
    'Name': 'Lenovo Yoga',
    'Category': 'Laptop',
    'SerialNo': 'CB27932009',
    'InvoiceNo': 'INV-2878',
    'Status': 'Assigned'
  },
  {
    'TaskID': 2,
    'Name': 'Acer Aspire',

```

```

        'Category': 'Others',
        'SerialNo': 'CB35728290',
        'InvoiceNo': 'INV-3456',
        'Status': 'In-repair'
    },
    {
        'TaskID': 3,
        'Name': 'Apple MacBook',
        'Category': 'Laptop',
        'SerialNo': 'CB35628728',
        'InvoiceNo': 'INV-2763',
        'Status': 'In-repair'
    }
  ]];
  let columnData: ColumnsModel[] = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  function importMongo(): void {
    qryBldrObj.setMongoQuery('{"$and":[{"TaskID":1001},{
"$or":[{"Category":{"$regex":"Order"}}]}]}');
  }
  return (
    <div>
      <QueryBuilderComponent width='100%' dataSource={hardwareData}
        columns={columnData} ref={(scope) => { qryBldrObj = scope as
        QueryBuilderComponent; }}/>
      <div className="e-qb-button">
        <ButtonComponent id="importSql" cssClass='e-primary'
        content='Set MongoDB Rules' onClick = {importMongo}/>
      </div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />,document.getElementById('querybuilder'));
```

Exporting

Exporting from the Query Builder allows users to preserve or store the created conditions. The defined conditions can be exported using various methods, including:

Exporting to JSON Object

You can extract the established conditions in the Query Builder and convert them into a structured JSON object format using the [getRules](#) method. This process enables users to save or transfer the conditions for further use or analysis in other applications or systems that support JSON data.

Exporting to SQL Query

Exporting to SQL involves converting the defined conditions within the Query Builder into SQL queries. This functionality allows users to generate SQL code representing the conditions set in the Query Builder, which can then be executed directly on a SQL database or used for further analysis and

processing. SQL exporting supports various types, including Inline SQL, Parameter SQL, and Named Parameter SQL.

Exporting to Inline SQL Query

Exporting to Inline SQL Query entails embedding the defined conditions from the Query Builder directly into SQL statements within the exported code. This method ensures that the conditions are seamlessly integrated into the SQL query syntax, enabling straightforward execution or further processing within SQL database systems. This can be achieved using the [getSqlFromRules](#) method.

APP.JSX

```
{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DialogComponent } from '@syncfusion/ej2-react-popups';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
  let qryBldrObj;
  let dialogInstance;
  let animationSettings = { effect: 'Zoom', duration: 400, delay: 0 };
  let columnData = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  let importRules = {
    'condition': 'or',
    'rules': [{
      'field': 'Category',
      'label': 'Category',
      'operator': 'equal',
      'type': 'string',
      'value': 'Laptop'
    }]
  };
  function getSql() {
    dialogInstance.content =
      qryBldrObj.getSqlFromRules(qryBldrObj.getRules());
    dialogInstance.show();
  }
  function getRule() {
    const validRule = qryBldrObj.getValidRules(qryBldrObj.rule);
    dialogInstance.content = '<pre>' + JSON.stringify(validRule, null,
4) + '</pre>';
    dialogInstance.show();
  }
  return (<div>
    <QueryBuilderComponent width='100%' dataSource={hardwareData}
    ref={scope => { qryBldrObj = scope; }} columns={columnData}
    rule={importRules}/>
    <div className="e-qb-button">
```

```

        <ButtonComponent cssClass='e-primary' content='Get Sql'
onClick={getSql}/>
        <ButtonComponent cssClass='e-primary' content='Get Rule'
onClick={getRule}/>
    </div>
    <DialogComponent id="defaultdialog" showCloseIcon={true}
animationSettings={animationSettings} ref={dialog => dialogInstance =
dialog} height='auto' header='Querybuilder' visible={false} width='50%'/>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { AnimationSettingsModel, DialogComponent } from '@syncfusion/ej2-
react-popups';
import { ColumnsModel, QueryBuilderComponent, RuleModel } from
'@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
    let qryBldrObj: QueryBuilderComponent;
    let dialogInstance: DialogComponent;
    let animationSettings: AnimationSettingsModel = { effect: 'Zoom',
duration: 400, delay: 0 };
    let columnData: ColumnsModel[] = [
        { field: 'TaskID', label: 'Task ID', type: 'number' },
        { field: 'Name', label: 'Name', type: 'string' },
        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
    let importRules: RuleModel = {
        'condition': 'or',
        'rules': [{
            'field': 'Category',
            'label': 'Category',
            'operator': 'equal',
            'type': 'string',
            'value': 'Laptop'
        }]
    };
    function getSql(): void {
        dialogInstance.content =
qryBldrObj.getSqlFromRules(qryBldrObj.getRules());
        dialogInstance.show();
    }
    function getRule(): void {
        const validRule = qryBldrObj.getValidRules(qryBldrObj.rule);

```



```

        dialogInstance.content = '<pre>' + JSON.stringify(validRule, null,
4) + '</pre>';
        dialogInstance.show();
    }
    return (
        <div>
            <QueryBuilderComponent width='100%' dataSource={hardwareData}
ref={(scope) => { qryBldrObj = scope as QueryBuilderComponent; }}
columns={columnData} rule={importRules}/>
            <div className="e-qb-button">
                <ButtonComponent cssClass='e-primary' content='Get Sql'
onClick = {getSql}/>
                <ButtonComponent cssClass='e-primary' content='Get Rule'
onClick = {getRule}/>
            </div>
            <DialogComponent id="defaultdialog" showCloseIcon={true}
animationSettings={animationSettings} ref={dialog => dialogInstance = dialog
as DialogComponent} height='auto' header='Querybuilder' visible={false}
width='50%' />
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
{% enddraw %}

```

Exporting to Parameter SQL Query

Exporting to Parameter SQL involves incorporating the defined conditions from the Query Builder into SQL queries with parameters. This method allows for dynamic value assignment during execution, enhancing flexibility and adaptability in query processing within SQL database. This can be accomplished using the [getParameterizedSql](#) method for exporting to Parameter SQL query.

APP.JSX

```

import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { AnimationSettingsModel, DialogComponent } from '@syncfusion/ej2-react-popups';
QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
function App() {
    let qryBldrObj;
    let dialog;
    let columnData = [
        { field: 'TaskID', label: 'Task ID', type: 'number' },
        { field: 'Name', label: 'Name', type: 'string' },
        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
    let hardwareData = [{
        'TaskID': 1,
        'Name': 'Lenovo Yoga',

```

```

        'Category': 'Laptop',
        'SerialNo': 'CB27932009',
        'InvoiceNo': 'INV-2878',
        'Status': 'Assigned'
    },
    {
        'TaskID': 2,
        'Name': 'Acer Aspire',
        'Category': 'Others',
        'SerialNo': 'CB35728290',
        'InvoiceNo': 'INV-3456',
        'Status': 'In-repair'
    },
    {
        'TaskID': 3,
        'Name': 'Apple MacBook',
        'Category': 'Laptop',
        'SerialNo': 'CB35628728',
        'InvoiceNo': 'INV-2763',
        'Status': 'In-repair'
    }
    ]];
    let importRules = {
        'condition': 'or',
        'rules': [{
            'label': 'Category',
            'field': 'Category',
            'type': 'string',
            'operator': 'equal',
            'value': 'Laptop'
        }
    ]
    };

    function getSql() {
        dialog.content =
JSON.stringify(qryBldrObj.getParameterizedSql(qryBldrObj.getRules()), null,
2);
        dialog.show();
    }
    function getRule() {
        let validRule = qryBldrObj.getValidRules(qryBldrObj.rule);
        dialog.content = '<pre>' + JSON.stringify(validRule, null, 4) +
'</pre>';
        dialog.show();
    }
    return (<div>
        <QueryBuilderComponent width='100%' dataSource={hardwareData}
columns={columnData} rule={importRules} ref={(scope) => { qryBldrObj =
scope; }}/>
        <div className="e-qb-button">
            <ButtonComponent id="exportsql" cssClass='e-primary'
content='Get Parameter sql' onClick = {getSql}/>
            <ButtonComponent id="exportrule" cssClass='e-primary'
content='Get Rule' onClick = {getRule}/>
        </div>
        <DialogComponent id='dialog' width='50%'
animationSettings={animationSettings} header={"Query Builder"}

```

```

visible={false} closeOnEscape={false} showCloseIcon={true} ref={(scope) => {
  dialog = scope; }}></DialogComponent>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, QueryLibrary, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { AnimationSettingsModel, DialogComponent } from '@syncfusion/ej2-
react-popups';
QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
function App() {
  let qryBldrObj: QueryBuilderComponent;
  let dialog: DialogComponent;
  let animationSettings: AnimationSettingsModel;
  let hardwareData: Object[] = [{
    'TaskID': 1,
    'Name': 'Lenovo Yoga',
    'Category': 'Laptop',
    'SerialNo': 'CB27932009',
    'InvoiceNo': 'INV-2878',
    'Status': 'Assigned'
  },
  {
    'TaskID': 2,
    'Name': 'Acer Aspire',
    'Category': 'Others',
    'SerialNo': 'CB35728290',
    'InvoiceNo': 'INV-3456',
    'Status': 'In-repair'
  },
  {
    'TaskID': 3,
    'Name': 'Apple MacBook',
    'Category': 'Laptop',
    'SerialNo': 'CB35628728',
    'InvoiceNo': 'INV-2763',
    'Status': 'In-repair'
  }
  ];
  let columnData: ColumnsModel[] = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  let importRules: RuleModel = {
    'condition': 'or',

```

```

        'rules': [{
            'label': 'Category',
            'field': 'Category',
            'type': 'string',
            'operator': 'equal',
            'value': 'Laptop'
        }]
    };
    function getSql(): void {
        dialog.content =
JSON.stringify(qryBldrObj.getParameterizedSql(qryBldrObj.getRules()), null,
2);
        dialog.show();
    }
    function getRule(): void {
        let validRule: RuleModel =
qryBldrObj.getValidRules(qryBldrObj.rule);
        dialog.content = '<pre>' + JSON.stringify(validRule, null, 4) +
'</pre>';
        dialog.show();
    }
    return (
        <div>
            <div>
                <QueryBuilderComponent width='100%' ref={(scope) => { qryBldrObj
= scope as QueryBuilderComponent; }} dataSource={hardwareData}
columns={columnData} rule={importRules} />
                <div className="e-qb-button">
                    <ButtonComponent id="exportsql" cssClass='e-primary'
content='Get Parameter sql' onClick = {getSql}/>
                    <ButtonComponent id="exportrule" cssClass='e-primary'
content='Get Rule' onClick = {getRule}/>
                </div>
                <DialogComponent id='dialog' width='50%'
animationSettings={animationSettings} header={"Query Builder"}
visible={false} closeOnEscape={false} showCloseIcon={true} ref={(scope) => {
dialog = scope as DialogComponent; }}></DialogComponent>
            </div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Exporting to Named Parameter SQL Query

Exporting to Named Parameter SQL entails integrating the defined conditions from the Query Builder into SQL queries with named parameters. This method offers enhanced readability and flexibility during execution by using named placeholders for parameter values. Named Parameter SQL facilitates easier maintenance and modification of queries, making it convenient for dynamic parameter assignment within SQL database. This can be accomplished using the method [getParameterizedNamedSql](#) for exporting to Named Parameter SQL query.

APP.JSX

```

import { QueryBuilderComponent, QueryLibrary } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
function App() {
    let qryBldrObj;
    let dialog;
    let columnData = [
        { field: 'TaskID', label: 'Task ID', type: 'number' },
        { field: 'Name', label: 'Name', type: 'string' },
        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
    let hardwareData = [{
        'TaskID': 1,
        'Name': 'Lenovo Yoga',
        'Category': 'Laptop',
        'SerialNo': 'CB27932009',
        'InvoiceNo': 'INV-2878',
        'Status': 'Assigned'
    },
    {
        'TaskID': 2,
        'Name': 'Acer Aspire',
        'Category': 'Others',
        'SerialNo': 'CB35728290',
        'InvoiceNo': 'INV-3456',
        'Status': 'In-repair'
    },
    {
        'TaskID': 3,
        'Name': 'Apple MacBook',
        'Category': 'Laptop',
        'SerialNo': 'CB35628728',
        'InvoiceNo': 'INV-2763',
        'Status': 'In-repair'
    }
    ];
    let importRules = {
        'condition': 'or',
        'rules': [{
            'label': 'Category',
            'field': 'Category',
            'type': 'string',
            'operator': 'equal',
            'value': 'Laptop'
        }]
    };
    function getSql() {
        dialog.content =
        JSON.stringify(qryBldrObj.getParameterizedNamedSql(qryBldrObj.getRules()),
        null, 2);
    }
}

```

```

        dialog.show();
    }
    function getRule() {
        let validRule = qryBldrObj.getValidRules(qryBldrObj.rule);
        dialog.content = '<pre>' + JSON.stringify(validRule, null, 4) +
'</pre>';
        dialog.show();
    }
    return (<div>
        <QueryBuilderComponent width='100%' dataSource={hardwareData}
columns={columnData} rule={importRules} ref={(scope) => { qryBldrObj =
scope; }}/>
        <div className="e-qb-button">
            <ButtonComponent id="exportmongo" cssClass='e-primary'
content='Get Parameter Named sql' onClick = {getSql}/>
            <ButtonComponent id="exportrule" cssClass='e-primary'
content='Get Rule' onClick = {getRule}/>
        </div>
        <DialogComponent id='dialog' width='50%'
animationSettings={animationSettings} header={"Query Builder"}
visible={false} closeOnEscape={false} showCloseIcon={true} ref={(scope) => {
dialog = scope; }}></DialogComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, QueryLibrary, RuleModel } from
'@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { AnimationSettingsModel, DialogComponent } from '@syncfusion/ej2-
react-popups';
QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
function App() {
    let qryBldrObj: QueryBuilderComponent;
    let dialog: DialogComponent;
    let animationSettings: AnimationSettingsModel;
    let hardwareData: Object[] = [{
        'TaskID': 1,
        'Name': 'Lenovo Yoga',
        'Category': 'Laptop',
        'SerialNo': 'CB27932009',
        'InvoiceNo': 'INV-2878',
        'Status': 'Assigned'
    },
    {
        'TaskID': 2,
        'Name': 'Acer Aspire',
        'Category': 'Others',
        'SerialNo': 'CB35728290',
        'InvoiceNo': 'INV-3456',
    }
    ];
}

```

```

        'Status': 'In-repair'
    },
    {
        'TaskID': 3,
        'Name': 'Apple MacBook',
        'Category': 'Laptop',
        'SerialNo': 'CB35628728',
        'InvoiceNo': 'INV-2763',
        'Status': 'In-repair'
    }
    ]];
let columnData: ColumnsModel[] = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
];
let importRules: RuleModel = {
    'condition': 'or',
    'rules': [{
        'label': 'Category',
        'field': 'Category',
        'type': 'string',
        'operator': 'equal',
        'value': 'Laptop'
    }]
};
function getSql(): void {
    dialog.content =
JSON.stringify(qryBldrObj.getParameterizedNamedSql(qryBldrObj.getRules()),
null, 2);
    dialog.show();
}
function getRule(): void {
    let validRule: RuleModel =
qryBldrObj.getValidRules(qryBldrObj.rule);
    dialog.content = '<pre>' + JSON.stringify(validRule, null, 4) +
'</pre>';
    dialog.show();
}
return (
    <div>
        <QueryBuilderComponent width='100%' ref={(scope) => { qryBldrObj
= scope as QueryBuilderComponent; }} dataSource={hardwareData}
columns={columnData} rule={importRules} />
        <div className="e-qb-button">
            <ButtonComponent id="exportmongo" cssClass='e-primary'
content='Get Named Parameter Sql' onClick = {getSql}/>
            <ButtonComponent id="exportmongo" cssClass='e-primary'
content='Get Rule' onClick = {getRule}/>
        </div>
        <DialogComponent id='dialog' width='50%'
animationSettings={animationSettings} header={"Query Builder"}
visible={false} closeOnEscape={false} showCloseIcon={true} ref={(scope) => {
dialog = scope as DialogComponent; }}></DialogComponent>
    </div>

```

```

    );
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Exporting to MongoDB Query

Exporting to MongoDB Query involves converting the defined conditions within the Query Builder into MongoDB query syntax. This process allows users to generate MongoDB queries representing the conditions set in the Query Builder, which can then be executed directly on a MongoDB database or used for further analysis and processing. This can be accomplished using the [getMongoQuery](#) method for exporting to MongoDB query.

APP.JSX

```

import { QueryBuilderComponent, QueryLibrary } from '@syncfusion/ej2-react-querybuilder';
import { AnimationSettingsModel, DialogComponent } from '@syncfusion/ej2-react-popups';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
  let qryBldrObj;
  let dialog;
  let animationSettings;
  let columnData = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  let importRules = {
    'condition': 'or',
    'rules': [{
      'label': 'Category',
      'field': 'Category',
      'type': 'string',
      'operator': 'equal',
      'value': 'Laptop'
    }]
  };
  let hardwareData = [{
    'TaskID': 1,
    'Name': 'Lenovo Yoga',
    'Category': 'Laptop',
    'SerialNo': 'CB27932009',
    'InvoiceNo': 'INV-2878',
    'Status': 'Assigned'
  },
  {
    'TaskID': 2,

```



```

        'Name': 'Acer Aspire',
        'Category': 'Others',
        'SerialNo': 'CB35728290',
        'InvoiceNo': 'INV-3456',
        'Status': 'In-repair'
    },
    {
        'TaskID': 3,
        'Name': 'Apple MacBook',
        'Category': 'Laptop',
        'SerialNo': 'CB35628728',
        'InvoiceNo': 'INV-2763',
        'Status': 'In-repair'
    }
    ]];

    function getMongo() {
        dialog.content = qryBldrObj.getMongoQuery(qryBldrObj.getRules());
        dialog.show();
    }
    function getRule() {
        let validRule = qryBldrObj.getValidRules(qryBldrObj.rule);
        dialog.content = '<pre>' + JSON.stringify(validRule, null, 4) +
'</pre>';
        dialog.show();
    }
    return (<div>
        <QueryBuilderComponent width='100%' dataSource={hardwareData}
columns={columnData} rule={importRules} ref={(scope) => { qryBldrObj =
scope; }}/>
        <div className="e-qb-button">
            <ButtonComponent id="exportmongo" cssClass='e-primary'
content='Get MongoDB' onClick = {getMongo}/>
            <ButtonComponent id="exportrule" cssClass='e-primary'
content='Get Rule' onClick = {getRule}/>
        </div>
        <DialogComponent id='dialog' width='50%'
animationSettings={animationSettings} header={"Query Builder"}
visible={false} closeOnEscape={false} showCloseIcon={true} ref={(scope) => {
dialog = scope; }}></DialogComponent>
        </div>);
    }
    export default App;
    ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, QueryLibrary, RuleModel } from
'@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { AnimationSettingsModel, DialogComponent } from '@syncfusion/ej2-
react-popups';
QueryBuilderComponent.Inject(QueryLibrary);
// @ts-ignore
function App() {

```

```

let gryBldrObj: QueryBuilderComponent;
let dialog: DialogComponent;
let animationSettings: AnimationSettingsModel;
let hardwareData: Object[] = [{
  'TaskID': 1,
  'Name': 'Lenovo Yoga',
  'Category': 'Laptop',
  'SerialNo': 'CB27932009',
  'InvoiceNo': 'INV-2878',
  'Status': 'Assigned'
},
{
  'TaskID': 2,
  'Name': 'Acer Aspire',
  'Category': 'Others',
  'SerialNo': 'CB35728290',
  'InvoiceNo': 'INV-3456',
  'Status': 'In-repair'
},
{
  'TaskID': 3,
  'Name': 'Apple MacBook',
  'Category': 'Laptop',
  'SerialNo': 'CB35628728',
  'InvoiceNo': 'INV-2763',
  'Status': 'In-repair'
}
]];
let importRules: RuleModel = {
  'condition': 'or',
  'rules': [{
    'label': 'Category',
    'field': 'Category',
    'type': 'string',
    'operator': 'equal',
    'value': 'Laptop'
  }]
};
let columnData: ColumnsModel[] = [
  { field: 'TaskID', label: 'Task ID', type: 'number' },
  { field: 'Name', label: 'Name', type: 'string' },
  { field: 'Category', label: 'Category', type: 'string' },
  { field: 'SerialNo', label: 'Serial No', type: 'string' },
  { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
  { field: 'Status', label: 'Status', type: 'string' }
];
function getMongo(): void {
  dialog.content = gryBldrObj.getMongoQuery(gryBldrObj.getRules());
  dialog.show();
}
function getRule(): void {
  let validRule: RuleModel =
gryBldrObj.getValidRules(gryBldrObj.rule);
  dialog.content = '<pre>' + JSON.stringify(validRule, null, 4) +
'</pre>';
  dialog.show();
}
return (

```

```

    <div>
      <QueryBuilderComponent width='100%' ref={(scope) => { qryBldrObj
= scope as QueryBuilderComponent; }} dataSource={hardwareData}
rule={importRules} columns={columnData} />
      <div className="e-qb-button">
        <ButtonComponent id="exportmongo" cssClass='e-primary'
content='Get MongoDB' onClick = {getMongo}/>
        <ButtonComponent id="exportmongo" cssClass='e-primary'
content='Get Rule' onClick = {getRule}/>
      </div>
      <DialogComponent id='dialog' width='50%'
animationSettings={animationSettings} header={"Query Builder"}
visible={false} closeOnEscape={false} showCloseIcon={true} ref={(scope) => {
dialog = scope as DialogComponent; }}></DialogComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Lock Group/Rule in React Query builder component

The Query Builder provides the functionality to lock individual rules or entire groups. When a rule is locked, it prevents users from modifying its field, operator, and value, effectively disabling these components. Similarly, locking a group disables all elements contained within it. This feature offers users greater control over their query configurations, ensuring that specific rules or groups remain unchanged. Additionally, users can manage the visibility of locking buttons through the [showButtons](#) function, allowing for seamless control over the locking mechanism.

You can lock groups and rules by interacting through the user interface and methods.

- Use the [lockGroup](#) method to lock group.
- Use [lockRule](#) method to lock rule.

APP.JSX

```

import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
  let qryBldrObj;
  let columnData = [
    { field: 'EmployeeID', label: 'Employee ID', operators: [{ key:
'Equal', value: 'equal' },
    { key: 'Greater than', value: 'greaterthan' }, { key: 'Less
than', value: 'lessthan' }], step: 10, type: 'number' },
    { field: 'FirstName', label: 'First Name', type: 'string' },
    { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'], },
    { field: 'Title', label: 'Title', type: 'string' },
    { field: 'HireDate', label: 'Hire Date', type: 'date', format:
'dd/MM/yyyy' },
    { field: 'Country', label: 'Country', type: 'string' },

```

```

    { field: 'City', label: 'City', type: 'string' }
  ];
  let importRules = {
    'condition': 'and',
    'rules': [{
      'field': 'EmployeeID',
      'label': 'Employee ID',
      'operator': 'equal',
      'type': 'number',
      'value': 1001
    },
    {
      'field': 'HireDate',
      'label': 'Hire Date',
      'operator': 'equal',
      'type': 'date',
      'value': '07/05/1991'
    },
    {
      'field': 'Title',
      'label': 'Title',
      'operator': 'equal',
      'type': 'string',
      'value': 'Sales Manager'
    }
  ]
};
function lockGroup() {
  qryBldrObj.lockGroup("querybuilder_group0");
}
function lockRule() {
  qryBldrObj.lockRule("querybuilder_group0_rule0");
}
return (<div>
  <QueryBuilderComponent id="querybuilder" width='100%'
dataSource={hardwareData} ref={ (scope) => { qryBldrObj = scope; }}
rule={importRules} columns={columnData} showButtons={{lockGroup: false,
lockRule: false}}/>
  <div className="e-qb-button">
    <ButtonComponent id="lockgroup" cssClass='e-primary'
content='Lock Group' onClick={lockGroup}/>
    <ButtonComponent id="lockrule" cssClass='e-primary'
content='Lock Rule' onClick={lockRule}/>
  </div>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, RuleModel } from
'@syncfusion/ej2-react-querybuilder';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore

```

```

import { employeeData } from '../datasource.ts';
function App() {
  let qryBldrObj: QueryBuilderComponent;
  let columnData: ColumnsModel[] = [
    { field: 'EmployeeID', label: 'Employee ID', type: 'number' },
    { field: 'FirstName', label: 'First Name', type: 'string' },
    { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
    { field: 'Title', label: 'Title', type: 'string' },
    { field: 'HireDate', label: 'Hire Date', type: 'date', format:
'dd/MM/yyyy' },
    { field: 'Country', label: 'Country', type: 'string' },
    { field: 'City', label: 'City', type: 'string' }
  ];
  let importRules: RuleModel = {
    'condition': 'and',
    'rules': [{
      'label': 'Employee ID',
      'field': 'EmployeeID',
      'type': 'number',
      'operator': 'equal',
      'value': 1001
    },
    {
      'label': 'Title',
      'field': 'Title',
      'type': 'string',
      'operator': 'equal',
      'value': 'Sales Manager'
    },
    {
      condition: "or", rules: [
        { 'label': 'Title',
          'field': 'Title',
          'type': 'string',
          'operator': 'equal',
          'value': 'Engineer' }
      ]
    }
  ]
};
function lockGroup(): void {
  qryBldrObj.lockGroup("querybuilder_group0");
}
function lockRule(): void {
  qryBldrObj.lockRule("querybuilder_group0_rule0");
}
return (<div>
  <QueryBuilderComponent id="querybuilder" width='100%'
dataSource={employeeData} ref={(scope) => { qryBldrObj = scope as
QueryBuilderComponent; }} columns={columnData} rule={importRules}
showButtons={{lockGroup: false, lockRule: false}} />
  <div className="e-qb-button">
    <ButtonComponent id="lockgroup" cssClass='e-primary'
content='Lock Group' onClick = {lockGroup}/>
    <ButtonComponent id="lockrule" cssClass='e-primary'
content='Lock Rule' onClick = {lockRule}/>
  </div>
</div>);
}

```

```

        </div>
    </div>

    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Clone Group/Rule in React Query builder component

The Query Builder functionality extends to cloning both individual rules and entire groups. Utilizing the Clone options will generate an exact duplicate of a rule or group adjacent to the original one. This feature enables users to replicate complex query structures effortlessly. The [showButtons](#) function offers users the ability to toggle the visibility of these cloning buttons, providing convenient control over the cloning process within the Query Builder interface.

You can clone groups and rules by interacting through the user interface and methods.

- Use the [cloneGroup](#) method to clone group.
- Use [cloneRule](#) method to clone rule.

APP.JSX

```

import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
    let qryBldrObj;
    let columnData = [
        { field: 'EmployeeID', label: 'Employee ID', type: 'number' },
        { field: 'FirstName', label: 'First Name', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type: 'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'Hire Date', type: 'date', format: 'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules = {
        'condition': 'and',
        'rules': [{
            'label': 'Employee ID',
            'field': 'EmployeeID',
            'type': 'number',
            'operator': 'equal',
            'value': 1001
        },
        {
            'label': 'Title',
            'field': 'Title',
            'type': 'string',

```

```

        'operator': 'equal',
        'value': 'Sales Manager'
    },
    {
        condition: "or", rules: [
            { 'label': 'Title',
              'field': 'Title',
              'type': 'string',
              'operator': 'equal',
              'value': 'Engineer' }
        ]
    }
];
};
function cloneGroup() {
    qryBldrObj.cloneGroup("group0", "group1", 1);
}
function cloneRule() {
    qryBldrObj.cloneRule("group0_rule0", "group0", 1);
}
return (<div>
    <QueryBuilderComponent width='100%' dataSource={employeeData}
    ref={(scope) => { qryBldrObj = scope; }} rule={importRules}
    columns={columnData} showButtons={{cloneGroup: false, cloneRule: false}}/>
    <div className="e-qb-button">
        <ButtonComponent id="clongroup" cssClass='e-primary'
        content='Clone Group' onClick={cloneGroup}/>
        <ButtonComponent id="clonerule" cssClass='e-primary'
        content='Clone Rule' onClick={cloneRule}/>
    </div>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, RuleModel } from
'@syncfusion/ej2-react-querybuilder';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
    let qryBldrObj: QueryBuilderComponent;
    let columnData: ColumnsModel[] = [
        { field: 'EmployeeID', label: 'Employee ID', type: 'number' },
        { field: 'FirstName', label: 'First Name', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'Hire Date', type: 'date', format:
'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
}

```

```

];
let importRules: RuleModel = {
  'condition': 'and',
  'rules': [{
    'label': 'Employee ID',
    'field': 'EmployeeID',
    'type': 'number',
    'operator': 'equal',
    'value': 1001
  },
  {
    'label': 'Title',
    'field': 'Title',
    'type': 'string',
    'operator': 'equal',
    'value': 'Sales Manager'
  },
  {
    condition: "or", rules: [
      { 'label': 'Title',
        'field': 'Title',
        'type': 'string',
        'operator': 'equal',
        'value': 'Engineer' }
    ]
  }
]
};
function cloneGroup(): void {
  qryBldrObj.cloneGroup("group0", "group1", 1);
}
function cloneRule(): void {
  qryBldrObj.cloneRule("group0_rule0", "group0", 1);
}
return (
  <div>
    <QueryBuilderComponent width='100%' dataSource={employeeData}
    ref={(scope) => { qryBldrObj = scope as QueryBuilderComponent; }}
    columns={columnData} rule={importRules} showButtons={{cloneGroup: false,
    cloneRule: false}}/>
    <div className="e-qb-button">
      <ButtonComponent id="clonergroup" cssClass='e-primary'
      content='Clone Group' onClick = {cloneGroup}/>
      <ButtonComponent id="clonerule" cssClass='e-primary'
      content='Clone Rule' onClick = {cloneRule}/>
    </div>
  </div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```


Global local in React Query builder component

The **Localization** library allows you to localize default text content of the Query Builder. The Query Builder component has static text that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the locale value and translation object.

The following list of properties and its values are used in the Query Builder.

Locale key words	Text
-----	-----
AddGroup	Add Group
AddCondition	Add Condition
DeleteRule	Remove this condition
DeleteGroup	Delete group
Edit	EDIT
SelectField	Select a field
SelectOperator	Select operator
StartsWith	Starts With
EndsWith	Ends With
Contains	Contains
Equal	Equal
NotEqual	Not Equal
LessThan	Less Than
LessThanOrEqual	Less Than Or Equal
GreaterThan	Greater Than
GreaterThanOrEqual	Greater Than Or Equal
Between	Between
NotBetween	Not Between
In	In
NotIn	Not In
Remove	REMOVE
ValidationMessage	This field is required
SummaryViewTitle	Summary View
OtherFields	Other Fields
AND	AND
OR	OR
SelectValue	Enter Value

| IsEmpty | Is Empty |

| IsNotEmpty | Is Not Empty |

| IsNull | Is Null |

| IsNotNull | Is Not Null |

| True | True |

| False | False |

APP.JSX

```
import { L10n } from '@syncfusion/ej2-base';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
L10n.load({
  'de-DE': {
    'querybuilder': {
      "AddCondition": "Bedingung hinzufügen",
      "AddGroup": "Gruppe hinzufügen",
      "Between": "Zwischen",
      "Contains": "Enthält",
      "DeleteGroup": "Gruppe löschen",
      "DeleteRule": "Entfernen Sie diesen Zustand",
      "Edit": "BEARBEITEN",
      "EndsWith": "Endet mit",
      "Equal": "Gleich",
      "GreaterThan": "Größer als",
      "GreaterThanOrEqual": "Größer als oder gleich",
      "In": "Im",
      "LessThan": "Weniger als",
      "LessThanOrEqual": "Weniger als oder gleich",
      "NotBetween": "Nicht zwischen",
      "NotEqual": "Nicht gleich",
      "NotIn": "Nicht in",
      "Remove": "LÖSCHEN",
      "SelectField": "Wählen Sie ein Feld aus",
      "SelectOperator": "Operator auswählen",
      "StartsWith": "Beginnt mit",
      "ValidationMessage": "Dieses Feld wird benötigt",
      "True": "Wahr",
      "False": "Falsch",
    }
  }
});
function App() {
  let columnData = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
```

```

let importRules = {
  'condition': 'or',
  'rules': [{
    'field': 'Category',
    'label': 'Category',
    'operator': 'equal',
    'type': 'string',
    'value': 'Laptop'
  }]
};
return (<QueryBuilderComponent width='100%' locale='de-DE'
dataSource={hardwareData} columns={columnData} rule={importRules}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { L10n } from '@syncfusion/ej2-base';
import { ColumnsModel, QueryBuilderComponent, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
L10n.load({
  'de-DE': {
    'querybuilder': {
      "AddCondition": "Bedingung hinzufügen",
      "AddGroup": "Gruppe hinzufügen",
      "Between": "Zwischen",
      "Contains": "Enthält",
      "DeleteGroup": "Gruppe löschen",
      "DeleteRule": "Entfernen Sie diesen Zustand",
      "Edit": "BEARBEITEN",
      "EndsWith": "Endet mit",
      "Equal": "Gleich",
      "GreaterThan": "Größer als",
      "GreaterThanOrEqual": "Größer als oder gleich",
      "In": "Im",
      "LessThan": "Weniger als",
      "LessThanOrEqual": "Weniger als oder gleich",
      "NotBetween": "Nicht zwischen",
      "NotEqual": "Nicht gleich",
      "NotIn": "Nicht in",
      "Remove": "LÖSCHEN",
      "SelectField": "Wählen Sie ein Feld aus",
      "SelectOperator": "Operator auswählen",
      "StartsWith": "Beginnt mit",
      "ValidationMessage": "Dieses Feld wird benötigt",
      "True": "Wahr",
      "False": "Falsch",
    }
  }
});
function App() {

```

```

let columnData: ColumnsModel[] = [
  { field: 'TaskID', label: 'Task ID', type: 'number' },
  { field: 'Name', label: 'Name', type: 'string' },
  { field: 'Category', label: 'Category', type: 'string' },
  { field: 'SerialNo', label: 'Serial No', type: 'string' },
  { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
  { field: 'Status', label: 'Status', type: 'string' }
];
let importRules: RuleModel = {
  'condition': 'or',
  'rules': [{
    'field': 'Category',
    'label': 'Category',
    'operator': 'equal',
    'type': 'string',
    'value': 'Laptop'
  }]
};
return (
  <QueryBuilderComponent width='100%' locale='de-DE'
  dataSource={hardwareData} columns={columnData} rule={importRules} />
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Style and appearance in React Query builder component

To modify the QueryBuilder appearance, you need to override the default CSS of QueryBuilder component. Please find the list of CSS classes and its corresponding section in QueryBuilder component. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

CSS Class | Purpose of Class

- |.e-group-header .e-btn|To customize the condition button in querybuilder
- |.e-group-body .e-rule-container|To customize the querybuilder rule container
- |.e-group-container .e-group-header .e-dropdown-btn|To customize the querybuilder Add group/condition button
- |.e-query-builder .e-group-header .e-deletegroup|To customize the querybuilder Delete group button
- |.e-query-builder .e-rule-field .e-rule-value-delete .e-rule-delete|To customize the querybuilder Delete condition button
- |.e-query-builder .e-rule-list > ::after,.e-query-builder .e-rule-list > ::before|To customize the querybuilder group joining line
- |.e-query-builder .e-rule-container.e-joined-rule|To customize the querybuilder condition joining line

Accessibility in React Query Builder component

The Query Builder component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Query Builder component is outlined below.

| Accessibility Criteria | Compatibility |

```

| -- | -- |
| WCAG 2.2 Support |  |
| Section 508 Support |  |
| Screen Reader Support |  |
| Right-To-Left Support |  |
| Color Contrast |  |
| Mobile Device Support |  |
| Keyboard Navigation Support |  |
| Accessibility Checker Validation |  |
| Axe-core Accessibility Validation |  |
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>
<div> - All
features of the component meet the requirement.</div>
<div> - Some features of the component do not meet the requirement.</div>
<div> - The component does not meet the requirement.</div>

```

WAI-ARIA attributes

WAI-ARIA (Accessibility Initiative – Accessible Rich Internet Applications) defines a way to increase the accessibility of web pages, dynamic content, and user interface components developed with Ajax, HTML, JavaScript, and related technologies. ARIA provides additional semantics to describe the role, state, and functionality of web components.

The following list of ARIA attributes is used in Query Builder.

| Attributes | Purpose |

| --- | --- |

| **role** | Indicates the query builder component. |

Keyboard interaction

The Query Builder component followed the keyboard interaction guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Query Builder component.

| **Press** | **To do this** |

| --- | --- |

| **Tab / Shift + Tab** | **To focus the next item in the rule.** |

Ensuring accessibility

The Query Builder component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Query Builder component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Query Builder component with accessibility tools.

See also

- [Accessibility in Syncfusion React components](#)

How To

Render with rule in React Query builder component

You can render the QueryBuilder with the defined rules. For this, you should use the [rule](#) property.

APP.JSX

```
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
    let columnData = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules = {
        'condition': 'and',
        'rules': [{
            'field': 'EmployeeID',
            'label': 'EmployeeID',
            'operator': 'equal',
```

```

        'type': 'number',
        'value': 1001
    },
    {
        'field': 'Title',
        'label': 'Title',
        'operator': 'equal',
        'type': 'string',
        'value': 'Sales Manager'
    }
  ]
};
return (<QueryBuilderComponent width='100%' dataSource={employeeData}
columns={columnData} rule={importRules}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
    let columnData: ColumnsModel[] = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules: RuleModel = {
        'condition': 'and',
        'rules': [{
            'field': 'EmployeeID',
            'label': 'EmployeeID',
            'operator': 'equal',
            'type': 'number',
            'value': 1001
        },
        {
            'field': 'Title',
            'label': 'Title',
            'operator': 'equal',
            'type': 'string',
            'value': 'Sales Manager'
        }
    ]
    };
    return (

```

```

        <QueryBuilderComponent width='100%' dataSource={employeeData}
        columns={columnData}
            rule={importRules} />
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

In this sample, `datasource(employeeData)` has been referred from `datasource` typescript file.

Display mode in React Query builder component

Display options allow you to view the Query Builder in Vertically or Horizontally. For this, you should use the [displayMode](#) property.

APP.JSX

```

import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
    let columnData = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type: 'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format: 'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules = {
        'condition': 'and',
        'rules': [{
            'field': 'EmployeeID',
            'label': 'EmployeeID',
            'operator': 'equal',
            'type': 'number',
            'value': 1001
        },
        {
            'field': 'Title',
            'label': 'Title',
            'operator': 'equal',
            'type': 'string',
            'value': 'Sales Manager'
        }
    ]
    };
    return (<QueryBuilderComponent width='30%' dataSource={employeeData}
    columns={columnData} rule={importRules} displayMode='Vertical' />);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```


APP.TSX

```

import { QueryBuilderComponent, ColumnsModel, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { employeeData } from '../datasource.ts';
function App() {
    let columnData:ColumnsModel[] = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
        'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
        'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules: RuleModel = {
        'condition': 'and',
        'rules': [{
            'field': 'EmployeeID',
            'label': 'EmployeeID',
            'operator': 'equal',
            'type': 'number',
            'value': 1001
        },
        {
            'field': 'Title',
            'label': 'Title',
            'operator': 'equal',
            'type': 'string',
            'value': 'Sales Manager'
        }
    ]
    };
    return (
        <QueryBuilderComponent width='30%' dataSource={employeeData}
        columns={columnData} rule={importRules} displayMode='Vertical' />
    );
}
export default App;
ReactDOM.render(<App />,document.getElementById('querybuilder'));

```

The default view the query builder component is Horizontal.

The default view the query builder component in Vertical.

Sort columns in React Query builder component

SortDirection allows you to sort the columns bounded to the Query Builder to view the columns by ascending or descending order. You should set the [sortDirection](#) property to sort the fields.

APP.JSX

```

import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';

```

```

import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
  let columnData = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  let importRules = {
    'condition': 'or',
    'rules': [{
      'field': 'Category',
      'label': 'Category',
      'operator': 'equal',
      'type': 'string',
      'value': 'Laptop'
    },
    {
      'condition': 'and',
      'rules': [{
        'field': 'Status',
        'label': 'Status',
        'operator': 'notequal',
        'type': 'string',
        'value': 'Pending'
      },
      {
        'field': 'TaskID',
        'label': 'Task ID',
        'operator': 'equal',
        'type': 'number',
        'value': 5675
      }
    ]
  }
  ];
  return (<QueryBuilderComponent width='100%' sortDirection="Ascending"
    dataSource={hardwareData} columns={columnData} rule={importRules}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
  let columnData: ColumnsModel[] = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },

```

```

    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  let importRules: RuleModel = {
    'condition': 'or',
    'rules': [{
      'field': 'Category',
      'label': 'Category',
      'operator': 'equal',
      'type': 'string',
      'value': 'Laptop'
    },
    {
      'condition': 'and',
      'rules': [{
        'field': 'Status',
        'label': 'Status',
        'operator': 'notequal',
        'type': 'string',
        'value': 'Pending'
      },
      {
        'field': 'TaskID',
        'label': 'Task ID',
        'operator': 'equal',
        'type': 'number',
        'value': 5675
      }
    ]
  }
  ];
  return (
    <QueryBuilderComponent width='100%' sortDirection="Ascending"
    dataSource={hardwareData} columns={columnData} rule={importRules} />
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Restrict groups in React Query builder component

You can restrict the condition set by defining the [maxGroupCount](#) property. By default, the value is 5. In the below demo, the [maxGroupCount](#) is set to 2.

APP.JSX

```

import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
  let columnData = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
  ];

```

```

        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
    let importRules = {
        'condition': 'or',
        'rules': [{
            'field': 'Category',
            'label': 'Category',
            'operator': 'equal',
            'type': 'string',
            'value': 'Laptop'
        }]
    };
    return (<QueryBuilderComponent width='100%' maxGroupCount={2}
    dataSource={hardwareData} columns={columnData} rule={importRules}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { ColumnsModel, QueryBuilderComponent, RuleModel } from
'@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
    let columnData: ColumnsModel[] = [
        { field: 'TaskID', label: 'Task ID', type: 'number' },
        { field: 'Name', label: 'Name', type: 'string' },
        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
    let importRules: RuleModel = {
        'condition': 'or',
        'rules': [{
            'field': 'Category',
            'label': 'Category',
            'operator': 'equal',
            'type': 'string',
            'value': 'Laptop'
        }]
    };
    return (
        <QueryBuilderComponent width='100%' maxGroupCount={2}
        dataSource={hardwareData} columns={columnData} rule={importRules} />
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

You can use this property in the mobile mode to restrict the nested group creation.

State persistence in React Query builder component

State persistence allows you to maintain the current state in the browser's `localStorage` even if the browser is refreshed or if you move to the next page within the browser. State persistence stores the Query Builder's [rule](#) object in the local storage when the [enablePersistence](#) is defined to true.

APP.JSX

```
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
    let columnData = [
        { field: 'TaskID', label: 'Task ID', type: 'number' },
        { field: 'Name', label: 'Name', type: 'string' },
        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
    let importRules = {
        'condition': 'or',
        'rules': [{
            'field': 'Category',
            'label': 'Category',
            'operator': 'equal',
            'type': 'string',
            'value': 'Laptop'
        }]
    };
    return (<QueryBuilderComponent width='100%' enablePersistence={true}
    dataSource={hardwareData} columns={columnData} rule={importRules}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));
```

APP.TSX

```
import { ColumnsModel, QueryBuilderComponent, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
function App() {
    let columnData: ColumnsModel[] = [
        { field: 'TaskID', label: 'Task ID', type: 'number' },
        { field: 'Name', label: 'Name', type: 'string' },
        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
};
```

```

let importRules: RuleModel = {
  'condition': 'or',
  'rules': [{
    'field': 'Category',
    'label': 'Category',
    'operator': 'equal',
    'type': 'string',
    'value': 'Laptop'
  }]
};
return (
  <QueryBuilderComponent width='100%' enablePersistence={true}
  dataSource={hardwareData} columns={columnData} rule={importRules} />
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Rtl in React Query builder component

RTL provides an option to switch the text direction and layout of the Query Builder component from right to left. It improves the user experiences and accessibility for users who use right-to-left languages (Arabic, Farsi, Urdu, etc.). To enable RTL, set the [enableRtl](#) to true.

APP.JSX

```

import { L10n } from '@syncfusion/ej2-base';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
L10n.load({
  'ar-AE': {
    'querybuilder': {
      'AddCondition': 'إضافة الشرط',
      'AddGroup': 'إضافة مجموعة',
      'Between': 'ما بين',
      'Contains': 'يحتوي على',
      'DeleteGroup': 'حذف المجموعة',
      'DeleteRule': 'أزل هذا الشرط',
      'Edit': 'تصحيح',
      'EndsWith': 'ينتهي مع',
      'Equal': 'مساو',
      'GreaterThan': 'أكثر من',
      'GreaterThanOrEqual': 'أكبر من أو يساوي',
      'In': 'في',
      'LessThan': 'أقل من',
      'LessThanOrEqual': 'اصغر من أو يساوي',
      'NotBetween': 'ليس بينهما',
      'NotEqual': 'ليس متساوي',
      'NotIn': 'ليس في',
      'Remove': 'إزالة',
      'SelectField': 'اختر مجال',
      'SelectOperator': 'حدد المشغل',
      'StartsWith': 'ابدا ب',
      'ValidationMessage': 'هذه الخانة مطلوبة'
    }
  }
});

```

```

    }
  }
});
function App() {
  let columnData = [
    { field: 'TaskID', label: 'Task ID', type: 'number' },
    { field: 'Name', label: 'Name', type: 'string' },
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'SerialNo', label: 'Serial No', type: 'string' },
    { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
    { field: 'Status', label: 'Status', type: 'string' }
  ];
  let importRules = {
    'condition': 'or',
    'rules': [{
      'field': 'Category',
      'label': 'Category',
      'operator': 'equal',
      'type': 'string',
      'value': 'Laptop'
    }]
  };
  return (<QueryBuilderComponent width='100%' locale='ar-AE'
enableRtl={true} dataSource={hardwareData} columns={columnData}
rule={importRules}/>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import { L10n } from '@syncfusion/ej2-base';
import { ColumnsModel, QueryBuilderComponent, RuleModel } from
 '@syncfusion/ej2-react-querybuilder';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// @ts-ignore
import { hardwareData } from '../datasource.ts';
L10n.load({
  'ar-AE': {
    'querybuilder': {
      'AddCondition': 'إضافة الشرط',
      'AddGroup': 'إضافة مجموعة',
      'Between': 'ما بين',
      'Contains': 'يحتوي على',
      'DeleteGroup': 'حذف المجموعة',
      'DeleteRule': 'أزل هذا الشرط',
      'Edit': 'تصحيح',
      'EndsWith': 'ينتهي مع',
      'Equal': 'مساو',
      'GreaterThan': 'أكبر من',
      'GreaterThanOrEqual': 'أكبر من أو يساوي',
      'In': 'في',
      'LessThan': 'أقل من',
      'LessThanOrEqual': 'أصغر من أو يساوي',
      'NotBetween': 'ليس بينهما',

```

```

        'NotEqual': 'ليس متساوي',
        'NotIn': 'ليس في',
        'Remove': 'إزالة',
        'SelectField': 'اختر مجال',
        'SelectOperator': 'حدد المشغل',
        'StartsWith': 'ابدا ب',
        'ValidationMessage': 'هذه الخانة مطلوبة'
    }
}
});
function App() {
    let columnData: ColumnsModel[] = [
        { field: 'TaskID', label: 'Task ID', type: 'number' },
        { field: 'Name', label: 'Name', type: 'string' },
        { field: 'Category', label: 'Category', type: 'string' },
        { field: 'SerialNo', label: 'Serial No', type: 'string' },
        { field: 'InvoiceNo', label: 'Invoice No', type: 'string' },
        { field: 'Status', label: 'Status', type: 'string' }
    ];
    let importRules: RuleModel = {
        'condition': 'or',
        'rules': [{
            'field': 'Category',
            'label': 'Category',
            'operator': 'equal',
            'type': 'string',
            'value': 'Laptop'
        }]
    };
    return (
        <QueryBuilderComponent width='100%' locale='ar-AE' enableRtl={true}
        dataSource={hardwareData} columns={columnData} rule={importRules} />
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

Summary view in React Query builder component

Summary view allows you to show or hide the filtered query. By default, the value is false. You can enable by setting the [summaryView](#) property to true.

APP.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { QueryBuilderComponent } from '@syncfusion/ej2-react-querybuilder';
import { employeeData } from '../datasource.ts';
function App() {
    let columnData = [
        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
        'boolean', values: ['Mr.', 'Mrs.'],
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
        'dd/MM/yyyy' },

```



```

        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules = {
        'condition': 'and',
        'rules': [{
            'label': 'EmployeeID',
            'field': 'EmployeeID',
            'type': 'number',
            'operator': 'notequal',
            'value': '5'
        },
        {
            'condition': 'or',
            'rules': [{
                'label': 'Title Of Courtesy',
                'field': 'TitleOfCourtesy',
                'type': 'string',
                'operator': 'equal',
                'value': 'Mr.'
            },
            {
                'label': 'Country',
                'field': 'Country',
                'type': 'string',
                'operator': 'equal',
                'value': 'USA'
            }
        ],
        {
            'condition': 'and',
            'rules': [{
                'label': 'City',
                'field': 'City',
                'type': 'string',
                'operator': 'equal',
                'value': 'London'
            }
        ]
    }
    ];
    return (<QueryBuilderComponent width='30%' dataSource={employeeData}
    columns={columnData} rule={importRules}
    summaryView="true"></QueryBuilderComponent>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

APP.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { QueryBuilderComponent, ColumnsModel, RuleModel } from
'@syncfusion/ej2-react-querybuilder';
import { employeeData } from '../datasource.ts';
function App() {
    let columnData: ColumnsModel[] = [

```

```

        { field: 'EmployeeID', label: 'EmployeeID', type: 'number' },
        { field: 'FirstName', label: 'FirstName', type: 'string' },
        { field: 'TitleOfCourtesy', label: 'Title Of Courtesy', type:
'boolean', values: ['Mr.', 'Mrs.'] },
        { field: 'Title', label: 'Title', type: 'string' },
        { field: 'HireDate', label: 'HireDate', type: 'date', format:
'dd/MM/yyyy' },
        { field: 'Country', label: 'Country', type: 'string' },
        { field: 'City', label: 'City', type: 'string' }
    ];
    let importRules: RuleModel = {
        'condition': 'and',
        'rules': [{
            'label': 'EmployeeID',
            'field': 'EmployeeID',
            'type': 'number',
            'operator': 'notequal',
            'value': '5'
        },
        {
            'condition': 'or',
            'rules': [{
                'label': 'Title Of Courtesy',
                'field': 'TitleOfCourtesy',
                'type': 'string',
                'operator': 'equal',
                'value': 'Mr.'
            },
            {
                'label': 'Country',
                'field': 'Country',
                'type': 'string',
                'operator': 'equal',
                'value': 'USA'
            },
            {
                'condition': 'and',
                'rules': [{
                    'label': 'City',
                    'field': 'City',
                    'type': 'string',
                    'operator': 'equal',
                    'value': 'London'
                }]
            }
        ]
    }
    return (
        <QueryBuilderComponent width='30%' dataSource={employeeData}
columns={columnData} rule={importRules} summaryView="true"
></QueryBuilderComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('querybuilder'));

```

RadioButton

Getting Started

This section explains how to create a simple RadioButton, and configure its available functionalities in React using React quickstart application.

Dependencies

The following list of dependencies are required to use the RadioButton component in your application.

```
`javascript
|-- @syncfusion/ej2-react-buttons
|-- @syncfusion/ej2-react-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
`,`
```

Installation and Configuration

You can use [Create-react-app](#) to setup the applications. To install `create-react-app` run the following command.

```
`bash
npm install -g create-react-app
`,`
```

To set-up a React application in TypeScript environment, run the following command.

```
`bash
npx create-react-app my-app --template typescript
cd my-app
npm start
`,`
```

To set-up a React application in JavaScript environment, run the following command.

```
`bash
npx create-react-app my-app
cd my-app
npm start
`,`
```

Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) public registry.

To install RadioButton component, use the following command

```
`bash
npm install @syncfusion/ej2-react-buttons --save
```

,

Adding CSS Reference

Import the RadioButton component's required CSS references as follows in `src/App.css`.

```
`css
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
,
```

Adding RadioButton component to the Application

To include the RadioButton component in your application import the `RadioButtonComponent` from `ej2-react-buttons` package in `App.tsx`.

Add the RadioButton component in application as shown in below code example.

```
`ts
// Import the RadioButton.
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import './App.css';
// To render RadioButton.
function App() {
  return (<div style={{marginTop: '150px'}}>
    <RadioButtonComponent label="default" />
  </div>);
}
export default App;
,
```

Run the application

Run the application in the browser using the following command:

,

```
npm start
```

,

The following example shows a basic RadioButton component.

APP.JSX

```
import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
```

```
// To render RadioButton.
function App() {
  return (
    <ul>
      <li><RadioButtonComponent label="Option 1" name="default"/></li>
      <li><RadioButtonComponent label="Option 2" name="default"/></li>
    </ul>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));
```

APP.TSX

```
import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
// To render RadioButton.
function App() {
  return (
    <ul>
      <li><RadioButtonComponent label="Option 1" name="default" /></li>
      <li><RadioButtonComponent label="Option 2" name="default" /></li>
    </ul>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));
```

You can refer to our [React Radio Button](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Radio Button Example](#) that shows you how to render the Radio Button in React.

Label and size in React Radio button component

This section explains the different sizes and labels.

Label

RadioButton caption can be defined by using the [label](#) property. This reduces the manual addition of label for RadioButton. You can customize the label position before or after the RadioButton through the [labelPosition](#) property.

APP.JSX

```
import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
function App() {
  return (
    <ul>
      { /* Label position - Left. */ }
      <li><RadioButtonComponent label="Left Side Label" name="position"
labelPosition="Before"/></li>
      { /* Label position - Right. */ }
    </ul>
  );
}
```

```

        <li><RadioButtonComponent label="Right Side Label" name="position"
checked={true}/></li>
      </ul>;
    }
    export default App;
    ReactDOM.render(<App />, document.getElementById('radio-button'));

```

APP.TSX

```

import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
function App() {
  return (
    <ul>
      { /* Label position - Left. */ }
      <li><RadioButtonComponent label="Left Side Label" name="position"
labelPosition="Before" /></li>
      { /* Label position - Right. */ }
      <li><RadioButtonComponent label="Right Side Label" name="position"
checked={true} /></li>
    </ul>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));

```

Size

The different RadioButton sizes available are default and small. To reduce the size of the default RadioButton to small, set the [cssClass](#) property to `e-small`.

APP.JSX

```

import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
function App() {
  return <ul>
    { /* Small RadioButton. */ }
    <li><RadioButtonComponent label="Small" name="size" cssClass="e-
small"/></li>
    { /* Default RadioButton. */ }
    <li><RadioButtonComponent label="Default" name="size"/></li>
  </ul>;
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));

```

APP.TSX

```

import { enableRipple } from '@syncfusion/ej2-base';

```

```
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
function App() {
  return (
    <ul>
      { /* Small RadioButton. */ }
      <li><RadioButtonComponent label="Small" name="size" cssClass="e-small" /></li>
      { /* Default RadioButton. */ }
      <li><RadioButtonComponent label="Default" name="size" /></li>
    </ul>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));
```

See Also

- [How to customize the RadioButton appearance](#)

Accessibility in React RadioButton component

The RadioButton component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the RadioButton component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

| Screen Reader Support | |

| Right-To-Left Support | |

| Color Contrast | |

| Mobile Device Support | |

| Keyboard Navigation Support | |

| [Accessibility Checker](#) Validation | |

```
| Axe-core Accessibility Validation |  |
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>
<div> - All
features of the component meet the requirement.</div>
<div> - Some features of the component do not meet the requirement.</div>
<div> - The component does not meet the requirement.</div>
```

WAI-ARIA attributes

The RadioButton component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the RadioButton component:

Attributes	Purpose
------------	---------

---	---
-----	-----

aria-disabled	Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable.
---------------	--

Keyboard interaction

The RadioButton component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the RadioButton component.

Press	To do this
-------	------------

---	---
-----	-----

UP/Left arrow	Move and select the previous options.
---------------	---------------------------------------

Down/Right arrow	Move and select the next options.
------------------	-----------------------------------

Ensuring accessibility

The RadioButton component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the RadioButton component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the RadioButton component with accessibility tools.

See also

- [Accessibility in Syncfusion React components](#)

Style and appearance in React Radio button component

To modify the RadioButton appearance, you need to override the default CSS of RadioButton component. Please find the list of CSS classes and its corresponding section in RadioButton. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

CSS Class | Purpose of Class

- |.e-radio-wrapper|To customize the radio button wrapper
- |.e-radio + label:hover::before|To customize the radiobutton on hover
- |.e-radio:checked + label::after, e-radio:checked + label::before |To customize the checked radiobutton
- |.e-radio:checked:focus + label::before, .e-radio:checked + label:hover::before |To customize the checked radiobutton on hover

How To

Change radiobutton state in React Radio button component

The Essential JS 2 RadioButton contains 2 different states visually, they are as follows:

- Checked
- Unchecked

The RadioButton [checked](#) property is used to handle the checked and unchecked state. In the checked state an inner circle will be added to the visualization of RadioButton.

APP.JSX

```
import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
function App() {
    return (
        <ul>
            { /* checked state. */ }
            <li><RadioButtonComponent label="Option 1" name="state"
checked={true}/></li>
            { /* unchecked state. */ }
            <li><RadioButtonComponent label="Option 2" name="state"/></li>
        </ul>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));
```

APP.TSX

```
import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
function App() {
    return (
        <ul>
```

```

        {/* checked state. */}
        <li><RadioButtonComponent label="Option 1" name="state"
checked={true} /></li>
        {/* unchecked state. */}
        <li><RadioButtonComponent label="Option 2" name="state" /></li>
    </ul>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));

```

Customize radiobutton appearance in React Radio button component

You can customize the appearance of the RadioButton component by using the CSS rules. Define own CSS rules according to your requirement and assign the class name to the [cssClass](#) property.

The background and border color of the RadioButton is customized through the custom classes to create the primary, success, warning, danger, and info type of radio button.

APP.JSX

```

import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To customize RadioButton appearance.
function App() {
    return <ul>
        {/* Refer the 'e-primary' class details in 'style.css'. */}
        <li><RadioButtonComponent label="Primary" cssClass="e-primary"
name="custom"/></li>
        {/* Refer the 'e-success' class details in 'style.css'. */}
        <li><RadioButtonComponent label="Success" cssClass="e-success"
name="custom"/></li>
        {/* Refer the 'e-info' class details in 'style.css'. */}
        <li><RadioButtonComponent label="Info" cssClass="e-info"
checked={true} name="custom"/></li>
        {/* Refer the 'e-warning' class details in 'style.css'. */}
        <li><RadioButtonComponent label="Warning" cssClass="e-warning"
name="custom"/></li>
        {/* Refer the 'e-danger' class details in 'style.css'. */}
        <li><RadioButtonComponent label="Danger" cssClass="e-danger"
name="custom"/></li>
    </ul>;
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));

```

APP.TSX

```

import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To customize RadioButton appearance.
function App() {
    return (
        <ul>

```

```

        { /* Refer the 'e-primary' class details in 'style.css'. */ }
        <li><RadioButtonComponent label="Primary" cssClass="e-primary"
name="custom" /></li>
        { /* Refer the 'e-success' class details in 'style.css'. */ }
        <li><RadioButtonComponent label="Success" cssClass="e-success"
name="custom" /></li>
        { /* Refer the 'e-info' class details in 'style.css'. */ }
        <li><RadioButtonComponent label="Info" cssClass="e-info"
checked={true} name="custom" /></li>
        { /* Refer the 'e-warning' class details in 'style.css'. */ }
        <li><RadioButtonComponent label="Warning" cssClass="e-warning"
name="custom" /></li>
        { /* Refer the 'e-danger' class details in 'style.css'. */ }
        <li><RadioButtonComponent label="Danger" cssClass="e-danger"
name="custom" /></li>
    </ul>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));

```

Name and value in form submit in React Radio button component

The [name](#) attribute of the RadioButton is used to group RadioButton. When the RadioButton are grouped in form, the checked items [value](#) attribute will be post to server on form submit that can be retrieved through the name. The disabled and unchecked RadioButton value will not be sent to the server on form submit.

In the following code snippet, Credit / Debit card is in checked state. Now, the value that is in checked state will be sent on form submit.

APP.JSX

```

import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent, ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
// Name and Value attribute in form submit.
function App() {
    return (<form>
        <ul>
            <li><RadioButtonComponent name="payment" value="credit/debit"
label="Credit /Debit card" checked={true}/></li>
            <li><RadioButtonComponent name="payment" value="netbanking"
label="Net Banking"/></li>
            <li><RadioButtonComponent name="payment" value="cashondelivery"
label="Cash On Delivery"/></li>
            <li><RadioButtonComponent name="payment" value="others"
label="Others"/></li>
            <li><ButtonComponent
isPrimary={true}>Submit</ButtonComponent></li>
        </ul>
    </form>);
}
export default App;

```

```
ReactDOM.render(<App />, document.getElementById('radio-button'));
```

APP.TSX

```
import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent, ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
// Name and Value attribute in form submit.
function App() {
  return (
    <form>
      <ul>
        <li><RadioButtonComponent name="payment" value="credit/debit"
label="Credit /Debit card" checked={true} /></li>
        <li><RadioButtonComponent name="payment" value="netbanking"
label="Net Banking" /></li>
        <li><RadioButtonComponent name="payment" value="cashondelivery"
label="Cash On Delivery" /></li>
        <li><RadioButtonComponent name="payment" value="others"
label="Others" /></li>
        <li><ButtonComponent
isPrimary={true}>Submit</ButtonComponent></li>
      </ul>
    </form>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));
```

Right to left in React Radio button component

RadioButton component has RTL support. This can be achieved by setting [enableRtl](#) as `true`.

The following example illustrates how to enable right-to-left support in RadioButton component.

APP.JSX

```
import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
// To render RadioButton.
function App() {
  return <ul>
    <li><RadioButtonComponent label="Option 1" name="default"
enableRtl={true}/></li>
    <li><RadioButtonComponent label="Option 2" name="default"
enableRtl={true}/></li>
  </ul>;
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));
```

APP.TSX

```
import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
// To render RadioButton.
function App() {
    return (
        <ul>
            <li><RadioButtonComponent label="Option 1" name="default"
enableRtl={true} /></li>
            <li><RadioButtonComponent label="Option 2" name="default"
enableRtl={true} /></li>
        </ul>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));
```

Set the disabled state in React Radio button component

RadioButton component can be enabled/disabled by giving [disabled](#) property. To disable RadioButton component,

the `disabled` property can be set as `true`.

The following example illustrates how to disable a radio button and the selected one is displayed using [change](#) event.

APP.JSX

```
import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
// To render RadioButton.
function App() {
    let radioInstance1;
    let radioInstance2;
    let radioInstance3;
    function changeOption1() {
        document.getElementById('text').innerText = 'Selected : ' +
radioInstance1.label;
    }
    function changeOption2() {
        document.getElementById('text').innerText = 'Selected : ' +
radioInstance2.label;
    }
    function changeOption3() {
        document.getElementById('text').innerText = 'Selected : ' +
radioInstance3.label;
    }
    return <ul>
        <li><div id="text">Selected : Option 1</div></li>
```

```

    <li><RadioButtonComponent label="Option 1" name="default"
checked={true} change={changeOption1} ref={radio1 => radioInstance1 =
radio1}/></li>
    <li><RadioButtonComponent label="Option 2" name="default"
disabled={true} change={changeOption2} ref={radio2 => radioInstance2 =
radio2}/></li>
    <li><RadioButtonComponent label="Option 3" name="default"
change={changeOption3} ref={radio3 => radioInstance3 = radio3}/></li>
  </ul>;
}
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));

```

APP.TSX

```

import { enableRipple } from '@syncfusion/ej2-base';
import { RadioButtonComponent, ChangeEventArgs } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
// To render RadioButton.
function App() {
    let radioInstance1: RadioButtonComponent;
    let radioInstance2: RadioButtonComponent;
    let radioInstance3: RadioButtonComponent;
    function changeOption1(): void {
        (document.getElementById('text') as HTMLElement).innerText = 'Selected
: ' + radioInstance1.label;
    }

    function changeOption2(): void {
        (document.getElementById('text') as HTMLElement).innerText =
'Selected : ' + radioInstance2.label;
    }

    function changeOption3(): void {
        (document.getElementById('text') as HTMLElement).innerText =
'Selected : ' + radioInstance3.label;
    }

    return (
        <ul>
            <li><div id="text">Selected : Option 1</div></li>
            <li><RadioButtonComponent label="Option 1" name="default"
checked={true} change={changeOption1} ref={radio1 => radioInstance1 = radio1
as RadioButtonComponent}/></li>
            <li><RadioButtonComponent label="Option 2" name="default"
disabled={true} change={changeOption2} ref={radio2 => radioInstance2 =
radio2 as RadioButtonComponent}/></li>
            <li><RadioButtonComponent label="Option 3" name="default"
change={changeOption3} ref={radio3 => radioInstance3 = radio3 as
RadioButtonComponent}/></li>
        </ul>
    );
}

```

```
export default App;
ReactDOM.render(<App />, document.getElementById('radio-button'));
```

Ej1 api migration in React Radio button component

This article describes the API migration process of RadioButton component from Essential JS 1 to Essential JS 2.

Properties

{% raw %}

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| Label | **Property:** *text*

 <EJ.RadioButton id="radio" text="RadioButton"></EJ.RadioButton> | **Property:** *label*

 <RadioButtonComponent id="radio" label="RadioButton"></RadioButtonComponent> |

| Checked state | **Property:** *checked*

 <EJ.RadioButton id="radio" text="RadioButton" checked={true}></EJ.RadioButton> | **Property:** *checked*

 <RadioButtonComponent id="radio" label="RadioButton" checked={true}></RadioButtonComponent> |

| Adding custom css class | **Property:** *cssClass*

 <EJ.RadioButton id="radio" text="RadioButton" cssClass="custom-class"></EJ.RadioButton> | **Property:** *cssClass*

 <RadioButtonComponent id="radio" label="RadioButton" cssClass="custom-class"></RadioButtonComponent> |

| Disabled state | **Property:** *enabled*

 <EJ.RadioButton id="radio" text="RadioButton" enabled={false}></EJ.RadioButton> | **Property:** *disabled*

 <RadioButtonComponent id="radio" label="RadioButton" disabled={true}></RadioButtonComponent> |

| State persistence | **Property:** *enablePersistence*

 <EJ.RadioButton id="radio" text="RadioButton" enablePersistence={true}></EJ.RadioButton> | **Property:** *enablePersistence*

 <RadioButtonComponent id="radio" label="RadioButton" enablePersistence={true}></RadioButtonComponent> |

| RTL | **Property:** *enableRTL*

 <EJ.RadioButton id="radio" text="RadioButton" enableRTL={true}></EJ.RadioButton> | **Property:** *enableRtl*

 <RadioButtonComponent id="radio" label="RadioButton" enableRtl={true}></RadioButtonComponent> |

| HTML Attributes | **Property:** *htmlAttributes*

 var attributes = { required:"required" };
 <EJ.RadioButton id="radio" text="RadioButton" htmlAttributes={attributes}></EJ.RadioButton> | Not applicable |

| Id property | **Property:** *id*

 <EJ.RadioButton id="sync"></EJ.RadioButton> | Not applicable |

| Prefix value of Id | **Property:** *idPrefix*

 <EJ.RadioButton id="radio" idPrefix="react"></EJ.RadioButton> | Not applicable |

| Name attribute | **Property:** *name*

 <EJ.RadioButton id="radio" name="gender" checked={true} text="Male"></EJ.RadioButton> | **Property:** *name*

`<RadioButtonComponent id="radio" name="gender" checked={true} label="Male"></RadioButtonComponent>` |

| Value attribute | **Property:** *value* `

` `<EJ.RadioButon id="radio" name="gender" checked={true} value="male" text="Male"></EJ.RadioButon>` | **Property:** *value* `

` `<RadioButtonComponent id="radio" name="gender" checked={true} value="male" label="Male"></RadioButtonComponent>` |

| Size | **Property:** *size* `

` `<EJ.RadioButon id="radio" size="small" text="RadioButton"></EJ.RadioButon>` | **Property:** *cssClass* `

` `<RadioButtonComponent id="radio" label="RadioButton" cssClass="e-small"></RadioButtonComponent>` |

| Validation rules | **Property:** *validationRules* `

` `var rules = { required: true };
` `<EJ.RadioButon id="radio" text="RadioButton" validationRules={rules}></EJ.RadioButon>` |

Not applicable |

| Validation message | **Property:** *validationMessage* `

` `var rules = { required: true };
` `var message = { required: "Required RadioButton value" };
` `<EJ.RadioButon id="radio" text="RadioButton" validationRules={rules} validationMessage={message}></EJ.RadioButon>` |

Not applicable |

Methods

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| Destroy | **Method:** *destroy* `

` `<EJ.RadioButon id="radio" text="RadioButton"></EJ.RadioButon>` `
` `var radioButton = $('#radio').ejRadioButon('instance');` `
` `radioButton.destroy();` | **Method:** *destroy* `

` `<RadioButtonComponent id="radio" label="RadioButton" ref={(scope) => {this.radioButton = scope}}></RadioButtonComponent>` `
` `constructor(props: {}) {
` ` ` `this.radioButton.destroy();
` `}` |

| Disable the RadioButton | **Method:** *disable* `

` `<EJ.RadioButon id="radio" text="RadioButton"></EJ.RadioButon>` `
` `var radioButton = $('#radio').ejRadioButon('instance');` `
` `radioButton.disable();` | Not applicable |

| Enable the RadioButton | **Method:** *enable* `

` `<EJ.RadioButon id="radio" text="RadioButton"></EJ.RadioButon>` `
` `var radioButton = $('#radio').ejRadioButon('instance');` `
` `radioButton.enable();` | Not applicable |

Events

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| BeforeChange Event | **Event:** *beforeChange* `

` `<EJ.RadioButon id="radio" text="RadioButton" beforeChange={beforeChange}></EJ.RadioButon>` `
` `function beforeChange(args) {
` ` ` ` ` ` ` ` ` `/ code block */
` `}` | Not applicable |

| Change Event | **Event:** *change* `

` `<EJ.RadioButon id="radio" text="RadioButton" change={change}></EJ.RadioButon>` `
` `function change(args) {
` ` ` ` ` ` ` ` ` `/ code block /
` `}` | **Event:** *change* `

`


```

<RadioButtonComponent id="radio" label="RadioButton"
change={this.change.bind(this)}></RadioButtonComponent> <br/> change(args) {<br/>
&#160;&#160;&#160;&#160;/ code block / <br/>} |

| Create Event | Event: create <br/><br/> <EJ.RadioButton id="radio" text="RadioButton"
create={create}></EJ.RadioButton> <br/>function create(args) {<br/> &#160;&#160;&#160;&#160;/
code block / <br/>} | Event: created <br/><br/> <RadioButtonComponent id="radio"
label="RadioButton" created={this.created.bind(this)}></RadioButtonComponent> <br/> created()
{<br/> &#160;&#160;&#160;&#160;/ code block / <br/>} |

| Destroy Event | Event: destroy <br/><br/> <EJ.RadioButton id="radio" text="RadioButton"
destroy={destroy}></EJ.RadioButton> <br/>function destroy(args) {<br/>
&#160;&#160;&#160;&#160;/ code block */ <br/>} | Not applicable |

{% endraw %}

```

Range Navigator

Getting Started

This section explains you the steps required to create a simple range navigator and demonstrate the basic usage of the range navigator control.

Dependencies

The list of minimum dependencies required to use an range navigator are follows:

```

`javascript
|-- @syncfusion/ej2-react-charts
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-pdf-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-svg-base
,

```

Installation and configuration

You can use [create-react-app](#) to setup the applications.

To install `create-react-app` run the following command.

```

,
npm install -g create-react-app
,

```

- To set-up a React application in TypeScript environment, run the following command.

`

```
create-react-app quickstart --template typescript
```

```
cd quickstart
```

```
npm start
```

`

- To set-up a React application in JavaScript environment, run the following command.

`

```
create-react-app quickstart
```

```
cd quickstart
```

```
npm start
```

`

- Install Syncfusion packages using below command.

`

```
npm install @syncfusion/ej2-react-charts --save
```

`

Add Range Navigator to the Project

Now, you can start adding range navigator component in the application.

For getting started, add the Chart component in `src/App.tsx` file using following code.

```
`ts
```

```
import * as React from 'react';
```

```
import {RangeNavigatorComponent} from '@syncfusion/ej2-react-charts';
```

```
function App() {
```

```
  return (<RangeNavigatorComponent></RangeNavigatorComponent>);
```

```
};
```

```
export default App;
```

`

Now run the `npm start` command in the console, it will run your application and open the browser window.

`

```
npm start
```

`

The below example shows a basic Range Navigator.

INDEX.JSX

```
import { RangeNavigatorComponent } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
class App extends React.Component {
  render() {
    return <RangeNavigatorComponent id="charts"/>;
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
```

INDEX.TSX

```
import { RangeNavigatorComponent} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
class App extends React.Component {
  render() {
    return <RangeNavigatorComponent id="charts" />
  }
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
```

Module Injection

Chart component are segregated into individual feature-wise modules. In order to use a particular feature, you need to inject its feature service in the AppModule. In the current application, we are going to modify the above basic chart to visualize sales data for a particular year. For this application we are going to use line series, tooltip, data label, category axis and legend feature of the chart. Please find the relevant feature service name and description as follows.

- **AreaSeries** - Inject this module in to **services** to use area series.
- **DateTime** - Inject this module in to **services** to use date time feature.
- **RangeTooltip** - Inject this module in to **services** to use tooltip feature.

These modules should be injected to the **services** section as follows,

```
`javascript
```

```
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent, RangeTooltip} from
 '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
function App() {
  return <RangeNavigatorComponent id='charts'>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
```

```
</RangeNavigatorComponent>
```

```
};
```

```
export default App;
```

```
ReactDOM.render(<App />, document.getElementById("charts"));
```

```
`
```

Populate Range Navigator with Data

Now, we are going to provide data to the range navigator. Add a [series](#) object to the range navigator by using series property. Now map the field names x and y in the JSON data to the [xName](#) and [yName](#) properties of the series, then set the JSON data to dataSource property.

Since the JSON contains Datetime data, set the [valueType](#) range Navigator to **Category**. By default, the axis [valueType](#) is **Numeric**.

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-
01')]}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  return <RangeNavigatorComponent id='charts'
```

```

    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}]}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

Enable Tooltip

The tooltip is useful to show the selected data. You can enable the tooltip by setting the `enable` property as `true` in tooltip object and by injecting `RangeTooltip` module into the `services`.

INDEX.JSX

```

{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true, displayMode: 'Always' };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {

```

```

const data: object[] = bitCoinData;
const tooltip: RangeTooltipSettingsModel = { enable: true, displayMode:
'Always' };
return <RangeNavigatorComponent id='charts'
  valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
  tooltip={tooltip}>
  <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
  <RangenavigatorSeriesCollectionDirective>
    <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
  </RangenavigatorSeriesCollectionDirective>
</RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

Selecting range in React Range navigator component

The Range Selector's left and right thumbs are used to indicate the selected range in the large collection of data. A range can be selected in the following ways:

- By dragging the thumbs.
- By tapping on the labels.
- By setting the start and the end through the [value](#) property.

INDEX.JSX

```

{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
  tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
  </>
}
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

Lightweight in React Range navigator component

By default, when the [dataSource](#) for [series](#) is empty, a lightweight Range Selector will be shown without Chart.

INDEX.JSX

```
{% raw %}
import { DateTime, Inject, RangeNavigatorComponent } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { GetDateTimeData } from 'default-data.ts';
function App() {
    const data = GetDateTimeData(new Date(2018, 0, 1), new Date(2019, 0, 1));
    return <RangeNavigatorComponent id='charts' valueType='DateTime'
intervalType='Months' labelFormat='MMM' value={[new Date('2018-04-1'), new
Date('2018-08-1')]} dataSource={data} xName='x' yName='y'>
        <Inject services={[DateTime]} />
    </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{ % raw %}
```

```
import { DateTime, Inject, RangeNavigatorComponent } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { GetDateTimeData } from 'default-data.ts';
function App() {
  const data: object[] = GetDateTimeData(new Date(2018, 0, 1), new Date(2019, 0, 1));
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' intervalType='Months' labelFormat='MMM'
    value={[new Date('2018-04-1'), new Date('2018-08-1')]}
    dataSource={data} xName='x' yName='y'>
    <Inject services={[DateTime]} />
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

See Also

- [Period Selector](#)

Series types in React Range navigator component

To render the data, the Range Selector supports three types of series.

<!-- markdownlint-disable MD036 -->

Line

<!-- markdownlint-disable MD036 -->

To render a line series, use series [type](#) as **Line** and inject the **LineSeries** module using **RangeNavigator.Inject(LineSeries)** method. By default, the line series is rendered in the Range Selector.

INDEX.JSX

```
{% raw %}
import { LineSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
tooltip={tooltip}>
  <Inject services={[LineSeries, DateTime, RangeTooltip]} />
  <RangenavigatorSeriesCollectionDirective>
    <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Line' width={2} />
  </RangenavigatorSeriesCollectionDirective>

```



```

    </RangeNavigatorComponent>;
  };
  export default App;
  ReactDOM.render(<App />, document.getElementById("charts"));
  {% endraw %}

```

INDEX.TSX

```

{% raw %}
import {
  LineSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    tooltip={tooltip}>
    <Inject services={[LineSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Line' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
  </>
  export default App;
  ReactDOM.render(<App />, document.getElementById("charts"));
  {% endraw %}

```

Area

To render an area series, use series [type](#) as **Area** and inject **AreaSeries** module using **RangeNavigator.Inject(AreaSeries)** method.

INDEX.JSX

```

{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
  RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
  tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
  </>
  export default App;
  ReactDOM.render(<App />, document.getElementById("charts"));
  {% endraw %}

```

```

    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

StepLine

To render a Step line series, use series [type](#) as **Step Line** and inject **StepLineSeries** module using **RangeNavigator.Inject(StepLineSeries)** method.

INDEX.JSX

```

{% raw %}
import { StepLineSeries, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
  RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default-data.ts';
function App() {
  const data = double;
  const tooltip = { enable: true };

```

```

    return <RangeNavigatorComponent id='charts' labelPosition='Outside'
    tooltip={tooltip} value={[12, 30]}>
      <Inject services={[RangeTooltip, StepLineSeries]} />
      <RangenavigatorSeriesCollectionDirective>
        <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'>
          </RangenavigatorSeriesDirective>
        </RangenavigatorSeriesCollectionDirective>
      </RangeNavigatorComponent>;
    };
    export default App;
    ReactDOM.render(<App />, document.getElementById("charts"));
    {% endraw %}

```

INDEX.TSX

```

{% raw %}
import {
  StepLineSeries, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default-data.ts';
function App() {
  const data: object[] = double;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    labelPosition='Outside'
    tooltip={tooltip}
    value={[12, 30]}>
    <Inject services={[RangeTooltip, StepLineSeries]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'>
        </RangenavigatorSeriesDirective>
      </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent>
  </>;
  export default App;
  ReactDOM.render(<App />, document.getElementById("charts"));
  {% endraw %}

```

<!-- markdownlint-disable MD036 -->

Data in React Range navigator component

Numeric

The numeric scale is used to represent the numeric values of data in a Range Selector. By default, the [valueType](#) of a Range Selector is **Double**.

INDEX.JSX

```

{% raw %}
import { StepLineSeries, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
  RangeTooltip } from '@syncfusion/ej2-react-charts';

```

```
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default-data.ts';
function App() {
  const data = double;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' labelPosition='Outside'
  tooltip={tooltip} value={[12, 30]}>
    <Inject services={[RangeTooltip, StepLineSeries]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'>
      </RangenavigatorSeriesDirective>
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  StepLineSeries, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default-data.ts';
function App() {
  const data: object[] = double;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
  labelPosition='Outside'
  tooltip={tooltip}
  value={[12, 30]}>
    <Inject services={[RangeTooltip, StepLineSeries]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'>
      </RangenavigatorSeriesDirective>
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

Range

The minimum and the maximum of the scale will be calculated automatically based on the provided data. It can be customized by using the [minimum](#), [maximum](#), and [interval](#) properties.

INDEX.JSX

```
{% raw %}
```

```

import { StepLineSeries, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default-data.ts';
function App() {
  const data = double;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' labelPosition='Outside'
tooltip={tooltip} value={[12, 30]}>
    <Inject services={[RangeTooltip, StepLineSeries]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'>
    </RangenavigatorSeriesDirective>
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import {
  StepLineSeries, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default-data.ts';
function App() {
  const data: object[] = double;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    labelPosition='Outside'
    tooltip={tooltip}
    value={[12, 30]}>
    <Inject services={[RangeTooltip, StepLineSeries]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'>
    </RangenavigatorSeriesDirective>
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

Label Format

The numeric labels can be formatted using the [labelFormat](#) property and it supports all the globalized formats.

INDEX.JSX

```
{% raw %}
import { StepLineSeries, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default-data.ts';
function App() {
  const data = double;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' labelPosition='Outside'
labelFormat='n1' tooltip={tooltip} value={[12, 30]}>
  <Inject services={[RangeTooltip, StepLineSeries]} />
  <RangenavigatorSeriesCollectionDirective>
    <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'>
    </RangenavigatorSeriesDirective>
  </RangenavigatorSeriesCollectionDirective>
</RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  StepLineSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default-data.ts';
function App() {
  const data: object[] = double;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    labelPosition='Outside'
    labelFormat='n1'
    tooltip={tooltip}
    value={[12, 30]}>
    <Inject services={[RangeTooltip, StepLineSeries]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'>
      </RangenavigatorSeriesDirective>
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

The following table describes the result of applying some commonly used label formats on numeric values.

<!-- markdownlint-disable MD033 -->

Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The Number is rounded to 1 decimal place
1000	n2	1000.00	The Number is rounded to 2 decimal place
1000	n3	1000.000	The Number is rounded to 3 decimal place
0.01	p1	1.0%	The Number is converted to percentage with 1 decimal place
0.01	p2	1.00%	The Number is converted to percentage with 2 decimal place
0.01	p3	1.000%	The Number is converted to percentage with 3 decimal place
1000	c1	\$1,000.0	The Currency symbol is appended to number and number is rounded to 1 decimal place
1000	c2	\$1,000.00	The Currency symbol is appended to number and number is rounded to 2 decimal place

Custom Label Format

The Range Selector also supports the Custom Label formats using the placeholders such as **{value}\$**, in which the value represents the axis label, e.g. 20\$.

INDEX.JSX

```
{% raw %}
import { StepLineSeries, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default-data.ts';
function App() {
  const data = double;
  return <RangeNavigatorComponent id='charts' labelPosition='Outside'
labelFormat='{value}$' value={[12, 30]}>
    <Inject services={[RangeTooltip, StepLineSeries]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'>
    </RangenavigatorSeriesDirective>
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  StepLineSeries, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default-data.ts';
function App() {
  const data: object[] = double;
  return <RangeNavigatorComponent id='charts'
    labelPosition='Outside'
    labelFormat='{value}$'
    value={[12, 30]}>
    <Inject services={[RangeTooltip, StepLineSeries]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'>
        </RangenavigatorSeriesDirective>
      </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent>
  </>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

Logarithmic Axis

```
<!-- markdownlint-disable MD033 -->
```

The Logarithmic supports the logarithmic scale, and it is used to visualize the data when the Range Selector has numerical values in both the lower (e.g.: 10-6) and the higher (e.g.: 106) orders of the magnitude.

INDEX.JSX

```
import { StepLineSeries, Logarithmic, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
  RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
class App extends React.Component {
  data = [];
  max = 100;
  chartLoad() {
    let i;
    for (i = 0; i < 100; i++) {
      this.data.push({
        x: Math.pow(10, i * 0.1),
        y: Math.floor(Math.random() * (80 - 30 + 1)) + 30
      });
    }
  }
  render() {
```



```

        this.chartLoad();
        return <RangeNavigatorComponent id='charts' labelPosition='Outside'
valueType='Logarithmic' interval={1} value={[4, 6]}>
    <Inject services={[StepLineSeries, Logarithmic, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
        <RangenavigatorSeriesDirective dataSource={this.data} xName='x'
yName='y' type='StepLine' width={2}>
            </RangenavigatorSeriesDirective>
        </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent>;
    }
}
;
ReactDOM.render(<App />, document.getElementById("charts"));

```

INDEX.TSX

```

import {
    StepLineSeries, Logarithmic, Inject, RangeNavigatorComponent,
    RangenavigatorSeriesCollectionDirective,
    RangenavigatorSeriesDirective, RangeTooltip
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default_data.ts';
class App extends React.Component<{}, {}> {
    public data: object[] = [];
    public max: number = 100;
    public chartLoad(): void {
        let i: number;
        for (i = 0; i < 100; i++) {
            this.data.push({
                x: Math.pow(10, i * 0.1),
                y: Math.floor(Math.random() * (80 - 30 + 1)) + 30
            });
        }
    }
    render() {
        this.chartLoad();
        return <RangeNavigatorComponent id='charts'
            labelPosition='Outside'
            valueType='Logarithmic'
            interval={1}
            value={[4, 6]}>
            <Inject services={[StepLineSeries, Logarithmic, RangeTooltip]} />
            <RangenavigatorSeriesCollectionDirective>
                <RangenavigatorSeriesDirective dataSource={this.data} xName='x'
yName='y'
                    type='StepLine' width={2}>
                    </RangenavigatorSeriesDirective>
                </RangenavigatorSeriesCollectionDirective>
            </RangeNavigatorComponent>
        </>
    }
};
ReactDOM.render(<App />, document.getElementById("charts"));

```

To use logarithmic scale, inject the `Logarithmic` module using the `RangeNavigator.Inject(Logarithmic)`

method, and then set the `valueType` to `Logarithmic`.

Range

The minimum and the maximum of the Range Selector will be calculated automatically based on the provided data. It can be customized by using the [minimum](#), [maximum](#), and [interval](#) properties.

INDEX.JSX

```
import { StepLineSeries, Logarithmic, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
class App extends React.Component {
  data = [];
  max = 100;
  chartLoad() {
    let i;
    for (i = 0; i < 100; i++) {
      this.data.push({
        x: Math.pow(10, i * 0.1),
        y: Math.floor(Math.random() * (80 - 30 + 1)) + 30
      });
    }
  }
  render() {
    this.chartLoad();
    return <RangeNavigatorComponent id='charts' labelPosition='Outside'
valueType='Logarithmic' interval={1} value={[4, 6]}>
      <Inject services={[StepLineSeries, Logarithmic, RangeTooltip]}/>
      <RangenavigatorSeriesCollectionDirective>
        <RangenavigatorSeriesDirective dataSource={this.data} xName='x'
yName='y' type='StepLine' width={2}>
          </RangenavigatorSeriesDirective>
        </RangenavigatorSeriesCollectionDirective>
      </RangeNavigatorComponent>;
  }
};
ReactDOM.render(<App />, document.getElementById("charts"));
```

INDEX.TSX

```
import {
  StepLineSeries, Logarithmic, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default_data.ts';
class App extends React.Component<{}, {}> {
  public data: object[] = [];
```

```

public max: number = 100;
public chartLoad(): void {
  let i: number;
  for (i = 0; i < 100; i++) {
    this.data.push({
      x: Math.pow(10, i * 0.1),
      y: Math.floor(Math.random() * (80 - 30 + 1)) + 30
    });
  }
}
render() {
  this.chartLoad();
  return <RangeNavigatorComponent id='charts'
    labelPosition='Outside'
    valueType='Logarithmic'
    interval={1}
    value={[4, 6]}>
    <Inject services={[StepLineSeries, Logarithmic, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={this.data} xName='x'
yName='y'
        type='StepLine' width={2}>
      </RangenavigatorSeriesDirective>
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
}
};
ReactDOM.render(<App />, document.getElementById("charts"));

```

Logarithmic Base

The Logarithmic Base can be customized using the [logBase](#) property. The default value of this property is 10.

INDEX.JSX

```

import { StepLineSeries, Logarithmic, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
class App extends React.Component {
  data = [];
  max = 100;
  chartLoad() {
    let i;
    for (i = 0; i < 100; i++) {
      this.data.push({
        x: Math.pow(10, i * 0.1),
        y: Math.floor(Math.random() * (80 - 30 + 1)) + 30
      });
    }
  }
  render() {
    this.chartLoad();
    return <RangeNavigatorComponent id='charts' labelPosition='Outside'
valueType='Logarithmic' logBase={2} value={[4, 6]}>

```

```

        <Inject services={[StepLineSeries, Logarithmic, RangeTooltip]}/>
        <RangenavigatorSeriesCollectionDirective>
            <RangenavigatorSeriesDirective dataSource={this.data} xName='x'
yName='y' type='StepLine' width={2}>
                </RangenavigatorSeriesDirective>
            </RangenavigatorSeriesCollectionDirective>
        </RangeNavigatorComponent>;
    }
}
;
ReactDOM.render(<App />, document.getElementById("charts"));

```

INDEX.TSX

```

import {
    StepLineSeries, Logarithmic, Inject, RangeNavigatorComponent,
    RangenavigatorSeriesCollectionDirective,
    RangenavigatorSeriesDirective, RangeTooltip
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default_data.ts';
class App extends React.Component<{}, {}> {
    public data: object[] = [];
    public max: number = 100;
    public chartLoad(): void {
        let i: number;
        for (i = 0; i < 100; i++) {
            this.data.push({
                x: Math.pow(10, i * 0.1),
                y: Math.floor(Math.random() * (80 - 30 + 1)) + 30
            });
        }
    }
    render() {
        this.chartLoad();
        return <RangeNavigatorComponent id='charts'
            labelPosition='Outside'
            valueType='Logarithmic'
            logBase={2}
            value={[4, 6]}>
            <Inject services={[StepLineSeries, Logarithmic, RangeTooltip]}/>
            <RangenavigatorSeriesCollectionDirective>
                <RangenavigatorSeriesDirective dataSource={this.data} xName='x'
yName='y'
                    type='StepLine' width={2}>
                    </RangenavigatorSeriesDirective>
                </RangenavigatorSeriesCollectionDirective>
            </RangeNavigatorComponent>
        </>
    }
};
ReactDOM.render(<App />, document.getElementById("charts"));

```

Date-time

The Range Selector supports the DateTime scale and displays the DateTime values as labels in the specified format.

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
```

```
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

Date-time Range navigator supports date-time scale and displays date-time values as a labels in the specified format.

Interval Customization

The DateTime intervals can be customized using the [interval](#) and the [intervalType](#) properties of the Range Selector. For example, if the [interval](#) is set to 2 and the [intervalType](#) is set to years, the interval will be considered to be 2 years.

DateTime supports the following interval types:

- Auto
- Years
- Quarter
- Months
- Weeks
- Days
- Hours
- Minutes

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
```

```

    AreaSeries, DateTime, Inject, RangeNavigatorComponent,
    RangenavigatorSeriesCollectionDirective,
    RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
    const data: object[] = bitCoinData;
    const tooltip: RangeTooltipSettingsModel = { enable: true };
    return <RangeNavigatorComponent id='charts'
        valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
        tooltip={tooltip}>
        <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
        <RangenavigatorSeriesCollectionDirective>
            <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
        </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent>
    </>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

Label Format

The [labelFormat](#) property is used to format and parse the date to all globalize format.

INDEX.JSX

```

{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
    const data = bitCoinData;
    const tooltip = { enable: true };
    return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='y/M/d' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
        tooltip={tooltip}>
        <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
        <RangenavigatorSeriesCollectionDirective>
            <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
        </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='y/M/d' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
</>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

The following table shows the results of applying some common DateTime formats to the [labelFormat](#) property.

<!-- markdownlint-disable MD033 -->

Label Value	Label Format Property Value	Result	Description
new Date(2000, 03, 10)	EEEE	Monday	The Date is displayed in day format
new Date(2000, 03, 10)	yMd	04/10/2000	The Date is displayed in month/date/year format
new Date(2000, 03, 10)	MMM	Apr	The Shorthand month for the date is displayed
new Date(2000, 03, 10)	hm	12:00 AM	Time of the date value is displayed as label
new Date(2000, 03, 10)	hms	12:00:00 AM	The Label is displayed in hours:minutes:seconds format

Period selector in React Range navigator component

The period selector allows to select a range with specified periods.

Periods

An array of objects that allows the users to specify pre-defined time intervals. The [interval](#) property specifies the count value of the button, and the [text](#) property specifies the text to be displayed on the button. The [intervaltype](#) property allows the users to customize the interval type, and it supports the following types:

- Auto
- Years
- Quarter
- Months
- Weeks
- Days
- Hours
- Minutes
- Seconds

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip, PeriodSelector } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  const periodselector = {
    position: 'Top',
    periods: [
      { text: '1M', interval: 1, intervalType: 'Months' },
      { text: '3M', interval: 3, intervalType: 'Months' },
      { text: '6M', interval: 6, intervalType: 'Months' }, { text: 'YTD' },
      { text: '1Y', interval: 1, intervalType: 'Years' },
      { text: '2Y', interval: 2, intervalType: 'Years', selected: true }, {
text: 'All' }
    ]
  };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
tooltip={tooltip} periodSelectorSettings={periodselector}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip, PeriodSelector]}
/>
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, PeriodSelector,
  RangeTooltipSettingsModel, PeriodSelectorSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  const periodselector: PeriodSelectorSettingsModel = {
    position: 'Top',
    periods: [
      { text: '1M', interval: 1, intervalType: 'Months' },
      { text: '3M', interval: 3, intervalType: 'Months' },
      { text: '6M', interval: 6, intervalType: 'Months' }, { text: 'YTD' },
      { text: '1Y', interval: 1, intervalType: 'Years' },
      { text: '2Y', interval: 2, intervalType: 'Years', selected: true }, {
text: 'All' }
    ]
  };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={ [new Date('2017-09-01'),
new Date('2018-02-01')] }
    tooltip={tooltip}
    periodSelectorSettings={periodselector}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip, PeriodSelector]}
  />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

To use the period selector feature, inject the **PeriodSelector** module using the **RangeNavigator.Inject(PeriodSelector)** method.

Positioning period selector

The **position** property allows the users to position the period selector at the **Top** or **Bottom**.

INDEX.JSX

```
{% raw %}
```

```

import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip, PeriodSelector } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  const periodselector = {
    position: 'Bottom',
    periods: [
      { text: '1M', interval: 1, intervalType: 'Months' },
      { text: '3M', interval: 3, intervalType: 'Months' },
      { text: '6M', interval: 6, intervalType: 'Months' }, { text: 'YTD' },
      { text: '1Y', interval: 1, intervalType: 'Years' },
      { text: '2Y', interval: 2, intervalType: 'Years', selected: true }, {
text: 'All' }
    ]
  };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
tooltip={tooltip} periodSelectorSettings={periodselector}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip, PeriodSelector]}
/>
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, PeriodSelector,
  RangeTooltipSettingsModel, PeriodSelectorSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  const periodselector: PeriodSelectorSettingsModel = {
    position: 'Bottom',
    periods: [
      { text: '1M', interval: 1, intervalType: 'Months' },
      { text: '3M', interval: 3, intervalType: 'Months' },
      { text: '6M', interval: 6, intervalType: 'Months' }, { text: 'YTD' },

```

```

        { text: '1Y', interval: 1, intervalType: 'Years' },
        { text: '2Y', interval: 2, intervalType: 'Years', selected: true }, {
text: 'All' }
    ]
};
return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    tooltip={tooltip}
    periodSelectorSettings={periodselector}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip, PeriodSelector]}
/>
    <RangenavigatorSeriesCollectionDirective>
        <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

Height

The [height](#) property allows the users to specify the height of the period selector. The default value of the height property is **43px**.

INDEX.JSX

```

{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip, PeriodSelector } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
    const data = bitCoinData;
    const tooltip = { enable: true };
    const periodselector = {
        position: 'Top',
        height: 65,
        periods: [
            { text: '1M', interval: 1, intervalType: 'Months' },
            { text: '3M', interval: 3, intervalType: 'Months' },
            { text: '6M', interval: 6, intervalType: 'Months' }, { text: 'YTD' },
            { text: '1Y', interval: 1, intervalType: 'Years' },
            { text: '2Y', interval: 2, intervalType: 'Years', selected: true }, {
text: 'All' }
        ]
    };
    return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
    tooltip={tooltip} periodSelectorSettings={periodselector}>
        <Inject services={[AreaSeries, DateTime, RangeTooltip, PeriodSelector]}
/>
        <RangenavigatorSeriesCollectionDirective>

```

```

        <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
</RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import {
    AreaSeries, DateTime, Inject, RangeNavigatorComponent,
    RangenavigatorSeriesCollectionDirective,
    RangenavigatorSeriesDirective, RangeTooltip, PeriodSelector,
    RangeTooltipSettingsModel, PeriodSelectorSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
    const data: object[] = bitCoinData;
    const tooltip: RangeTooltipSettingsModel = { enable: true };
    const periodselector: PeriodSelectorSettingsModel = {
        position: 'Top',
        height: 65,
        periods: [
            { text: '1M', interval: 1, intervalType: 'Months' },
            { text: '3M', interval: 3, intervalType: 'Months' },
            { text: '6M', interval: 6, intervalType: 'Months' }, { text: 'YTD' },
            { text: '1Y', interval: 1, intervalType: 'Years' },
            { text: '2Y', interval: 2, intervalType: 'Years', selected: true }, {
text: 'All' }
        ]
    };
    return <RangeNavigatorComponent id='charts'
        valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
        tooltip={tooltip}
        periodSelectorSettings={periodselector}>
        <Inject services={[AreaSeries, DateTime, RangeTooltip, PeriodSelector]}
/>
        <RangenavigatorSeriesCollectionDirective>
            <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
        </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

Visibility of range navigator

The [disableRangeSelector](#) property allows the users to display only the period selector and not the Range Selector.

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent, RangeTooltip,
PeriodSelector } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
    const data = bitCoinData;
    const tooltip = { enable: true };
    const periodselector = {
        position: 'Top',
        periods: [
            { text: '1M', interval: 1, intervalType: 'Months' },
            { text: '3M', interval: 3, intervalType: 'Months' },
            { text: '6M', interval: 6, intervalType: 'Months' }, { text:
'YTD' },
            { text: '1Y', interval: 1, intervalType: 'Years' },
            { text: '2Y', interval: 2, intervalType: 'Years', selected: true
}, { text: 'All' }
        ]
    };
    return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
dataSource={data} xName='x' yName='y' disableRangeSelector={true}
tooltip={tooltip} periodSelectorSettings={periodselector}>
        <Inject services={[AreaSeries, DateTime, RangeTooltip,
PeriodSelector]} />
    </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
    AreaSeries, DateTime, Inject, RangeNavigatorComponent, RangeTooltip,
PeriodSelector, RangeTooltipSettingsModel, PeriodSelectorSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
    const data: object[] = bitCoinData;
    const tooltip: RangeTooltipSettingsModel = { enable: true };
    const periodselector: PeriodSelectorSettingsModel = {
        position: 'Top',
        periods: [
            { text: '1M', interval: 1, intervalType: 'Months' },
```

```

    { text: '3M', interval: 3, intervalType: 'Months' },
    { text: '6M', interval: 6, intervalType: 'Months' }, { text: 'YTD' },
    { text: '1Y', interval: 1, intervalType: 'Years' },
    { text: '2Y', interval: 2, intervalType: 'Years', selected: true }, {
text: 'All' }
  ]
};
return <RangeNavigatorComponent id='charts'
  valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
  dataSource={data} xName='x' yName='y'
  disableRangeSelector={true}
  tooltip={tooltip}
  periodSelectorSettings={periodselector}>
  <Inject services={[AreaSeries, DateTime, RangeTooltip, PeriodSelector]}
/>
</RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% enddraw %}

```

See Also

- [LightWeight](#)

Labels in React Range navigator component

Multilevel labels

The multi-level labels for the Range Selector can be enabled by setting the [enableGrouping](#) property to **true**. This is restricted to the DateTime axis alone.

INDEX.JSX

```

import { DateTime, Inject, RangeNavigatorComponent } from '@syncfusion/ej2-
react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
class App extends React.Component {
  data = [];
  chartLoad() {
    let value = 0;
    let point = {};
    for (let j = 1; j < 1090; j++) {
      value += (Math.random() * 10 - 5);
      value = value < 0 ? Math.abs(value) : value;
      point = { x: new Date(2000, 0, j), y: value, z: value + 10 };
      this.data.push(point);
    }
  }
  render() {
    this.chartLoad();
    return <RangeNavigatorComponent id='charts' valueType='DateTime'
intervalType='Months' intervalType='Quarter' enableGrouping={true}

```

```

value={ [new Date('2001-01-01'), new Date('2002-01-01')] }
dataSource={this.data} xName='x' yName='y'>
    <Inject services={[DateTime]}/>
</RangeNavigatorComponent>;
}
}
;
ReactDOM.render(<App />, document.getElementById("charts"));

```

INDEX.TSX

```

import {
    AreaSeries, DateTime, Inject, RangeNavigatorComponent,
    RangenavigatorSeriesCollectionDirective,
    RangenavigatorSeriesDirective, RangeTooltip
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { GetDateTimeData } from 'default_data.ts';
class App extends React.Component<{}, {}> {
    public data: object[] = [];
    public chartLoad(): void {
        let value: number = 0; let point: object = {};
        for (let j: number = 1; j < 1090; j++) {
            value += (Math.random() * 10 - 5);
            value = value < 0 ? Math.abs(value) : value;
            point = { x: new Date(2000, 0, j), y: value, z: value + 10 };
            this.data.push(point);
        }
    }
    render() {
        this.chartLoad();
        return <RangeNavigatorComponent id='charts'
            valueType='DateTime' intervalType='Months'
            intervalType='Quarter' enableGrouping={true}
            value={ [new Date('2001-01-01'), new Date('2002-01-01')] }
            dataSource={this.data} xName='x' yName='y'>
            <Inject services={[DateTime]}/>
        </RangeNavigatorComponent>
    }
};
ReactDOM.render(<App />, document.getElementById("charts"));

```

Grouping

The multi-level labels can be grouped using the [groupBy](#) property with the following interval types:

- Auto
- Years
- Quarter
- Months
- Weeks
- Days
- Hours

- Minutes
- Seconds

INDEX.JSX

```
import { DateTime, Inject, RangeNavigatorComponent } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
class App extends React.Component {
  data = [];
  chartLoad() {
    let value = 0;
    let point = {};
    for (let j = 1; j < 1090; j++) {
      value += (Math.random() * 10 - 5);
      value = value < 0 ? Math.abs(value) : value;
      point = { x: new Date(2000, 0, j), y: value, z: value + 10 };
      this.data.push(point);
    }
  }
  render() {
    this.chartLoad();
    return <RangeNavigatorComponent id='charts' valueType='DateTime'
    intervalType='Months' intervalType='Quarter' enableGrouping={true}
    groupBy='Years' value={[new Date('2001-01-01'), new Date('2002-01-01')]}
    dataSource={this.data} xName='x' yName='y'>
      <Inject services={[DateTime]}/>
    </RangeNavigatorComponent>;
  }
}
ReactDOM.render(<App />, document.getElementById("charts"));
```

INDEX.TSX

```
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { GetDateTimeData } from 'default_data.ts';
class App extends React.Component<{}, {}> {
  public data: object[] = [];
  public chartLoad(): void {
    let value: number = 0; let point: object = {};
    for (let j: number = 1; j < 1090; j++) {
      value += (Math.random() * 10 - 5);
      value = value < 0 ? Math.abs(value) : value;
      point = { x: new Date(2000, 0, j), y: value, z: value + 10 };
      this.data.push(point);
    }
  }
  render() {
```

```

    this.chartLoad();
    return <RangeNavigatorComponent id='charts'
      valueType='DateTime' intervalType='Months'
      intervalType='Quarter'
      enableGrouping={true}
      groupBy='Years'
      value={[new Date('2001-01-01'), new Date('2002-01-01')]}
      dataSource={this.data} xName='x' yName='y'>
      <Inject services={[DateTime]} />
    </RangeNavigatorComponent>
  }
};
ReactDOM.render(<App />, document.getElementById("charts"));

```

Smart labels

The [labelIntersectAction](#) property is used to avoid overlapping of labels. The following code sample shows the setting of [labelIntersectAction](#) property to **Hide**.

INDEX.JSX

```

{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='y/M/d' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
tooltip={tooltip} labelIntersectAction='Hide'>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';

```

```
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='y/M/d' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    tooltip={tooltip} labelIntersectAction='Hide'>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent>
  </>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

Label positioning

By default, the labels can be placed outside the Range Selector. It can also be placed inside the Range Selector using the [labelPosition](#) property.

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelPosition='Inside' value={[new Date('2017-09-01'), new Date('2018-02-
01')]} tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
```

```

} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelPosition='Inside' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
    tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% enddraw %}

```

Labels customization

The font size, color, family, etc. can be customized using the [labelStyle](#) setting.

INDEX.JSX

```

import { DateTime, Inject, RangeNavigatorComponent } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
class App extends React.Component {
  data = [];
  max = 100;
  chartLoad() {
    let value = 0;
    let point = {};
    for (let j = 1; j < 1090; j++) {
      value += (Math.random() * 10 - 5);
      value = value < 0 ? Math.abs(value) : value;
      point = { x: new Date(2000, 0, j), y: value, z: value + 10 };
      this.data.push(point);
    }
  }
  render() {
    this.chartLoad();
    return <RangeNavigatorComponent id='charts' valueType='DateTime'
intervalType='Months' labelFormat='MMM' value={[new Date('2001-01-01'), new
Date('2002-01-01')]} dataSource={this.data} xName='x' yName='y'>
      <Inject services={[DateTime]} />
    </RangeNavigatorComponent>;
  }
}
;
ReactDOM.render(<App />, document.getElementById("charts"));

```

INDEX.TSX

```
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { GetDateTimeData } from 'default_data.ts';
class App extends React.Component<{}, {}> {
  public data: object[] = [];
  public max: number = 100;
  public chartLoad(): void {
    let value: number = 0; let point: object = {};
    for (let j: number = 1; j < 1090; j++) {
      value += (Math.random() * 10 - 5);
      value = value < 0 ? Math.abs(value) : value;
      point = { x: new Date(2000, 0, j), y: value, z: value + 10 };
      this.data.push(point);
    }
  }
  render() {
    this.chartLoad();
    return <RangeNavigatorComponent id='charts'
      valueType='DateTime' intervalType='Months' labelFormat='MMM'
      value={[new Date('2001-01-01'), new Date('2002-01-01')]}
      dataSource={this.data} xName='x' yName='y'>
      <Inject services={[DateTime]} />
    </RangeNavigatorComponent>
  }
};
ReactDOM.render(<App />, document.getElementById("charts"));
```

Grid tick in React Range navigator component

Grid line customization

The gridlines indicate axis divisions by drawing the chart plot. Gridlines include helpful cues to the user, particularly for large or complicated charts. The **width**, **color**, and **dashArray** of the major gridlines can be customized by using the [majorGridLines](#) setting.

INDEX.JSX

```
import { StepLineSeries, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
  RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
class App extends React.Component {
  data = [
    { xData: 10, yData: 35 }, { xData: 20, yData: 28 },
    { xData: 30, yData: 34 }, { xData: 40, yData: 32 },
    { xData: 50, yData: 40 }
  ];
  tooltip = { enable: true };
  majorgridLines = { width: 4, color: 'blue', dashArray: '5,5' };
};
```

```

render() {
    return <RangeNavigatorComponent id='charts' labelPosition='Outside'
    tooltip={this.tooltip} majorGridLines={this.majorgridLines} value={[25, 40]}>
        <Inject services={[RangeTooltip, StepLineSeries]} />
        <RangenavigatorSeriesCollectionDirective>
            <RangenavigatorSeriesDirective dataSource={this.data} xName='xData'
yName='yData'>
                </RangenavigatorSeriesDirective>
            </RangenavigatorSeriesCollectionDirective>
        </RangeNavigatorComponent>;
    }
;
ReactDOM.render(<App />, document.getElementById("charts"));

```

INDEX.TSX

```

import {
    StepLineSeries, DateTime, Inject, RangeNavigatorComponent,
    RangenavigatorSeriesCollectionDirective,
    RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default_data.ts';
class App extends React.Component<{}, {}> {
    public data: object[] = [
        { xData: 10, yData: 35 }, { xData: 20, yData: 28 },
        { xData: 30, yData: 34 }, { xData: 40, yData: 32 },
        { xData: 50, yData: 40 }
    ];
    public tooltip: RangeTooltipSettingsModel = { enable: true };
    public majorgridLines = { width: 4, color: 'blue', dashArray: '5,5' };
    render() {
        return <RangeNavigatorComponent id='charts'
            labelPosition='Outside'
            tooltip={this.tooltip}
            majorGridLines={this.majorgridLines}
            value={[25, 40]}>
            <Inject services={[RangeTooltip, StepLineSeries]} />
            <RangenavigatorSeriesCollectionDirective>
                <RangenavigatorSeriesDirective dataSource={this.data} xName='xData'
yName='yData'>
                    </RangenavigatorSeriesDirective>
                </RangenavigatorSeriesCollectionDirective>
            </RangeNavigatorComponent>
        }
    };
    ReactDOM.render(<App />, document.getElementById("charts"));
}

```

Tick line customization

Ticklines are the small lines which is drawn on the axis line representing the axis labels. Ticklines will be drawn outside the axis by default. The **width**, **color**, and **dashArray** of the major ticklines can be customized by using the [majorTickLines](#) setting.

INDEX.JSX

```

import { StepLineSeries, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
class App extends React.Component {
  data = [
    { xData: 10, yData: 35 }, { xData: 20, yData: 28 },
    { xData: 30, yData: 34 }, { xData: 40, yData: 32 },
    { xData: 50, yData: 40 }
  ];
  tooltip = { enable: true };
  majortickLines = { width: 3, color: 'red' };
  render() {
    return <RangeNavigatorComponent id='charts' labelPosition='Outside'
    tooltip={this.tooltip} majorTickLines={this.majortickLines} value={[25, 40]}>
      <Inject services={[RangeTooltip, StepLineSeries]} />
      <RangenavigatorSeriesCollectionDirective>
        <RangenavigatorSeriesDirective dataSource={this.data} xName='xData'
yName='yData'>
          </RangenavigatorSeriesDirective>
        </RangenavigatorSeriesCollectionDirective>
      </RangeNavigatorComponent>;
    }
  }
;
ReactDOM.render(<App />, document.getElementById("charts"));

```

INDEX.TSX

```

import {
  StepLineSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { double } from 'default_data.ts';
class App extends React.Component<{}, {}> {
  public data: object[] = [
    { xData: 10, yData: 35 }, { xData: 20, yData: 28 },
    { xData: 30, yData: 34 }, { xData: 40, yData: 32 },
    { xData: 50, yData: 40 }
  ];
  public tooltip: RangeTooltipSettingsModel = { enable: true };
  public majortickLines = { width: 3, color: 'red' };
  render() {
    return <RangeNavigatorComponent id='charts'
      labelPosition='Outside'
      tooltip={this.tooltip}
      majorTickLines={this.majortickLines}
      value={[25, 40]}>
      <Inject services={[RangeTooltip, StepLineSeries]} />
      <RangenavigatorSeriesCollectionDirective>

```

```

        <RangenavigatorSeriesDirective dataSource={this.data} xName='xData'
yName='yData'>
        </RangenavigatorSeriesDirective>
    </RangenavigatorSeriesCollectionDirective>
</RangeNavigatorComponent>
    }
};
ReactDOM.render(<App />, document.getElementById("charts"));

```

Customization in React Range navigator component

Navigator appearance

The Range Selector can be customized by using the [navigatorStyleSettings](#). The [selectedRegionColor](#) property is used to specify the color for the selected region, whereas the [unselectedRegionColor](#) property is used to specify the color for the unselected region.

INDEX.JSX

```

{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
    const data = bitCoinData;
    const tooltip = { enable: true };
    const navigatorstyleSettings = {
        unselectedRegionColor: 'skyblue',
        selectedRegionColor: 'pink'
    };
    return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}>
navigatorStyleSettings={navigatorstyleSettings} tooltip={tooltip}>
        <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
        <RangenavigatorSeriesCollectionDirective>
            <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
        </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import {
    AreaSeries, DateTime, Inject, RangeNavigatorComponent,
    RangenavigatorSeriesCollectionDirective,
    RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel,
    StyleSettingsModel
} from '@syncfusion/ej2-react-charts';

```



```

import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  const navigatorstyleSettings: StyleSettingsModel = {
    unselectedRegionColor: 'skyblue',
    selectedRegionColor: 'pink'
  };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    navigatorStyleSettings={navigatorstyleSettings}
    tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
  </>
  export default App;
  ReactDOM.render(<App />, document.getElementById("charts"));
  {% endraw %}

```

Thumb

The thumb property allows to customize the border, fill color, size, and type of thumb. Thumbs can be of two shapes: **Circle** and **Rectangle**.

INDEX.JSX

```

{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  const navigatorstyleSettings = {
    thumb: {
      type: 'Rectangle',
      border: { width: 2, color: 'red' },
      fill: 'pink'
    }
  };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
navigatorStyleSettings={navigatorstyleSettings} tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />

```

```

    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% enddraw %}

```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel,
  StyleSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  const navigatorstylesettings: StyleSettingsModel = {
    thumb: {
      type: 'Rectangle',
      border: { width: 2, color: 'red' },
      fill: 'pink'
    }
  };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    navigatorStyleSettings={navigatorstylesettings}
    tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
  </>
  export default App;
  ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

Border customization

Using the `navigatorBorder`, the `width` and `color` of the Range Selector border can be customized.

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
```

```
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  const border = { width: 4, color: 'green' };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
navigatorBorder={border} tooltip={tooltip}>
  <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
  <RangenavigatorSeriesCollectionDirective>
    <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
  </RangenavigatorSeriesCollectionDirective>
</RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  const border = { width: 4, color: 'green' };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    navigatorBorder={border} tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

Deferred update

If the [enableDeferredUpdate](#) property is set to **true**, then the changed event will be triggered after dragging the slider. If the [enableDeferredUpdate](#) is **false**, then the changed event will be triggered when dragging the slider. By default, the [enableDeferredUpdate](#) is set to **false**.

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

Allow snapping

The [allowSnapping](#) property toggles the placement of the slider exactly to the left or on the nearest interval.

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
allowSnapping={true} tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    allowSnapping={true} tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
```

```
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

Animation

The speed of the animation can be controlled using the [animationDuration](#) property. The default value of the [animationDuration](#) property is **500** milliseconds.

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
animationDuration={2000} tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
new Date('2018-02-01')]}
    animationDuration={2000} tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
  </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

```

    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

See Also

- [Grid and Tick Lines](#)
- [Labels](#)

Tool tip in React Range navigator component

<!-- markdownlint-disable MD036 -->

The tooltip for sliders are supported by the Range Selector. Sliders are used in the Range Selector to select data from a specific range. The tooltip displays the selected start and end values.

<!-- markdownlint-disable MD013 -->

Customization

Tooltip can be customized using the following properties:

- enable - Customizes the visibility of the tooltip.
- fill - Customizes the background color of the tooltip.
- opacity - Customizes the opacity of the tooltip.
- textStyle - Customizes the font size, color, family, style, weight, alignment, and overflow of the tooltip.

INDEX.JSX

```

{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true, displayMode: 'Always', fill: 'red',
opacity: 0.6, textStyle: { style: 'Italic', color: 'blue', size: '12px' } };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;

```

```
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true, displayMode:
    'Always', fill: 'red', opacity: 0.6, textStyle: { style: 'Italic', color:
    'blue', size: '12px' } };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'),
    new Date('2018-02-01')]}
    tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
    type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
  </>
  export default App;
  ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

Label Format

The **labelFormat** property in the tooltip is used to format and parse the date to all globalize formats.

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
  RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true, displayMode: 'Always', labelFormat: 'y/M/d'
  };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
  value={[new Date('2017-09-01'), new Date('2018-02-01')]} tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
```



```

    <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
  </RangenavigatorSeriesCollectionDirective>
</RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% enddraw %}

```

INDEX.TSX

```

{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true, displayMode:
'Always', labelFormat: 'y/M/d' };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' value={[new Date('2017-09-01'), new Date('2018-02-
01')]}
    tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent>
  </>;
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

The following table shows the results of applying some common date and time formats to the `labelFormat` property.

<!-- markdownlint-disable MD033 -->

Label Value	Label Format Property Value	Result	Description
<code>new Date(2000, 03, 10)</code>	<code>EEEE</code>	Monday	The Date is displayed in day format
<code>new Date(2000, 03, 10)</code>	<code>yMd</code>	04/10/2000	The Date is displayed in month/date/year format
<code>new Date(2000, 03, 10)</code>	<code>MMM</code>	Apr	The Shorthand month for the date is displayed

new Date(2000, 03, 10)	hm	12:00 AM	Time of the date value is displayed as label
new Date(2000, 03, 10)	hms	12:00:00 AM	The Label is displayed in hours:minutes:seconds format

Rtl in React Range navigator component

The Range Selector supports right-to-left (RTL), which can be enabled with the [enableRtl](#) property.

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  return <RangeNavigatorComponent id='charts' valueType='DateTime'
value={[new Date('2017-09-01'), new Date('2018-02-01')]} tooltip={tooltip}
enableRtl={true}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  return <RangeNavigatorComponent id='charts'
    valueType='DateTime' value={[new Date('2017-09-01'), new Date('2018-02-
01')]}
    tooltip={tooltip} enableRtl={true}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
  </RangeNavigatorComponent>;
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}
```

```

    <RangenavigatorSeriesCollectionDirective>
      <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
  </RangeNavigatorComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById("charts"));
{% endraw %}

```

Export print in React Range navigator component

Export

The rendered Range Selector can be exported to **JPEG, PNG, SVG, or PDF** format by using the [export](#) method in the Range Selector. This method contains the following parameters:

- **Type** - To specify the export type. The component can be exported to **JPEG, PNG, SVG, or PDF** format.
- **File name** - To specify the file name to export.
- **Orientation** - To specify the orientation type. This is applicable only for PDF export type.

INDEX.JSX

```

{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
  const tooltip = { enable: true };
  function clickHandler() {
    range.export('PNG', 'sample');
  }
  let range;
  return (<div>
    <ButtonComponent value='export'
onClick={clickHandler.bind(this)}>Export</ButtonComponent>
    <RangeNavigatorComponent id='charts' ref={g => range = g}
valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'), new
Date('2018-02-01')]} tooltip={tooltip}>
      <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
      <RangenavigatorSeriesCollectionDirective>
        <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
      </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent></div>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('charts'));
{% endraw %}

```

INDEX.TSX

```
{% raw %}
import {
  AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective,
  RangenavigatorSeriesDirective, RangeTooltip, RangeNavigator,
  RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { bitCoinData } from 'default-data.ts';
function App() {
  const data: object[] = bitCoinData;
  const tooltip: RangeTooltipSettingsModel = { enable: true };
  function clickHandler() {
    range.export('PNG', 'sample');
  }
  let range: RangeNavigator;
  return (<div>
    <ButtonComponent value='export'
onClick={clickHandler.bind(this)}>Export</ButtonComponent>
    <RangeNavigatorComponent id='charts' ref={g => range = g}
      valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'), new Date('2018-02-01')]}
      tooltip={tooltip}>
      <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
      <RangenavigatorSeriesCollectionDirective>
        <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
      </RangenavigatorSeriesCollectionDirective>
    </RangeNavigatorComponent></div>
  );
  export default App;
  ReactDOM.render(<App />, document.getElementById('charts'));
}{% endraw %}
```

Print

The rendered Range Selector can be printed directly from the browser by calling the public method [print](#).

INDEX.JSX

```
{% raw %}
import { AreaSeries, DateTime, Inject, RangeNavigatorComponent,
  RangenavigatorSeriesCollectionDirective, RangenavigatorSeriesDirective,
  RangeTooltip } from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { bitCoinData } from 'default-data.ts';
function App() {
  const data = bitCoinData;
```

```

const tooltip = { enable: true };
function clickHandler() {
    range.print();
}
let range;
return (<div> <ButtonComponent value='print'
onClick={clickHandler.bind(this)}>Print</ButtonComponent>
    <RangeNavigatorComponent id='charts' ref={g => range = g}
valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-01'), new
Date('2018-02-01')]} tooltip={tooltip}>
    <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
    <RangenavigatorSeriesCollectionDirective>
        <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
    </RangenavigatorSeriesCollectionDirective>
</RangeNavigatorComponent></div>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('charts'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import {
    AreaSeries, DateTime, Inject, RangeNavigatorComponent,
    RangenavigatorSeriesCollectionDirective,
    RangenavigatorSeriesDirective, RangeTooltip, RangeNavigator,
    RangeTooltipSettingsModel
} from '@syncfusion/ej2-react-charts';
import * as React from "react";
import * as ReactDOM from "react-dom";
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { bitCoinData } from 'default-data.ts';
function App() {
    const data: object[] = bitCoinData;
    const tooltip: RangeTooltipSettingsModel = { enable: true };
    function clickHandler() {
        range.print();
    }
    let range: RangeNavigator;
    return (<div> <ButtonComponent value='print'
onClick={clickHandler.bind(this)}>Print</ButtonComponent>
        <RangeNavigatorComponent id='charts' ref={g => range = g}
        valueType='DateTime' labelFormat='MMM-yy' value={[new Date('2017-09-
01'), new Date('2018-02-01')]}
        tooltip={tooltip}>
            <Inject services={[AreaSeries, DateTime, RangeTooltip]} />
            <RangenavigatorSeriesCollectionDirective>
                <RangenavigatorSeriesDirective dataSource={data} xName='x' yName='y'
type='Area' width={2} />
            </RangenavigatorSeriesCollectionDirective>
        </RangeNavigatorComponent></div>);
};
export default App;
ReactDOM.render(<App />, document.getElementById('charts'));

```

```
{% endraw %}
```

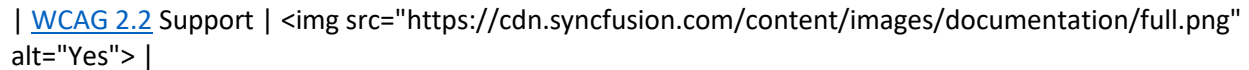
Accessibility in React Range navigator component

The Range navigator component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Range navigator component is outlined below.

| Accessibility Criteria | Compatibility |

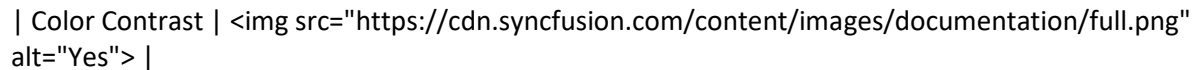
| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

WAI-ARIA attributes

The Range navigator component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Range navigator component:

- region (role)
- aria-label (attribute)

Keyboard interaction

The Range navigator component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Range navigator component.

| **Press** | **To do this** |

| --- | --- |

| **Tab** | Moves the focus to the Range navigator element. |

| **Ctrl + P** | Prints the Range navigator. |

Ensuring accessibility

The Range navigator component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Range navigator component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Range navigator component with accessibility tools.

[See also](#)

- [Accessibility in Syncfusion React components](#)

Ej1 api migration in React Range navigator component

This article describes the API migration process of Chart component from Essential JS 1 to Essential JS 2.

RangeNavigator

<!-- markdownlint-disable MD033 -->

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

|Allow snapping| **Property:** *allowSnapping*

<EJ.RangeNavigator
allowSnapping={true}>
</EJ.RangeNavigator>| **Property:** *allowSnapping*

<RangeNavigatorComponent id='charts' allowSnapping={true}>
</RangeNavigatorComponent>|

|Animation duration| Not Applicable| **Property:** *allowSnapping*

<RangeNavigatorComponent id='charts' animationDuration='2000'>
</RangeNavigatorComponent>|

|Border for range navigator| **Property:** *border*

var border = { color:'blue', width:2, opacity:0.5 };
<EJ.RangeNavigator
border ={border}>
</EJ.RangeNavigator>| **Property:**

navigatorBorder

<RangeNavigatorComponent id='charts' navigatorBorder={ width: 4, color: 'green'}>
</RangeNavigatorComponent>|

|dataSource for range navigator| **Property:** *dataSource*

var series = { type: 'line', dataSource: data };
<EJ.RangeNavigator
series={series}>
</EJ.RangeNavigator>| **Property:** *dataSource*

<RangenavigatorSeriesDirective dataSource={data}>
</RangenavigatorSeriesDirective>|

|enabling deferred update for range navigator| **Property:** *enableDeferredUpdate*

<EJ.RangeNavigator
enableDeferredUpdate={true}>
</EJ.RangeNavigator>| **Property:** *enableDeferredUpdate*

<RangeNavigatorComponent id='charts' enableDeferredUpdate={false}>
</RangeNavigatorComponent>|

|multilevel level labels| **Property:** *labelSettings.higherLevelLabels*

var labelSettings= { higherLevel: { } };
<EJ.RangeNavigator
labelSettings = {labelSettings}>
</EJ.RangeNavigator>| **Property:** *enableGrouping*

<RangeNavigatorComponent id='charts' enableGrouping={false}>
</RangeNavigatorComponent>|

|enabling scroll bar| **Property:** *enableScrollBar*

var labelSettings= { higherLevel: { } };
<EJ.RangeNavigator
enableScrollBar={true}>
</EJ.RangeNavigator>| Not Applicable|

|enabling auto resizing| **Property:** *enableAutoResize*

<EJ.RangeNavigator
enableAutoResize={true}>
</EJ.RangeNavigator>| Not Applicable|

|enabling isResponsive| **Property:** *isResponsive*

<EJ.RangeNavigator
isResponsive={true}>
</EJ.RangeNavigator>| Not Applicable|

|enabling RTL for range navigator| **Property:** *enableRtl*

<EJ.RangeNavigator
enableRtl={true}>
</EJ.RangeNavigator>| **Property:** *enableRtl*

<RangeNavigatorComponent id='charts' enableRtl={false}>
</RangeNavigatorComponent>|

|interval for range navigator| **Property:** *valueAxisSettings.range.interval*

var valueAxisSettings= {Interval: 5 };
<EJ.RangeNavigator
valueAxisSettings={valueAxisSettings}>
</EJ.RangeNavigator>| **Property:** *interval*

<RangeNavigatorComponent id='charts' interval={1}>
</RangeNavigatorComponent>|

|intervaltype for range navigator| **Property:** *valueAxisSettings.range.intervalType*

var valueAxisSettings= {intervalType: 'Years'};
<EJ.RangeNavigator
valueAxisSettings={valueAxisSettings}>
</EJ.RangeNavigator>| **Property:** *intervalType*

<RangeNavigatorComponent id='charts' intervalType='Months'>
</RangeNavigatorComponent>|

|labelformat for range navigator| Not applicable| **Property:** *labelFormat*

<RangeNavigatorComponent id='charts' labelFormat='yMd'>
</RangeNavigatorComponent>|

|label intersect action for range navigator| Not applicable| **Property:** *labelIntersectAction*

```
<br/><br/><RangeNavigatorComponent id='charts'
labelIntersectAction='Hide'><br/></RangeNavigatorComponent>|
```

|labelStyle range navigator| **Property:** *valueAxisSettings.font*

var valueAxisSettings= {font: { }
};
<EJ.RangeNavigator
valueAxisSettings={valueAxisSettings}>

```
<br/></EJ.RangeNavigator>| Property: labelStyle <br/><br/><RangeNavigatorComponent
id='charts' labelStyle='Hide'><br/></RangeNavigatorComponent>|
```

|locale of range navigator| **Property:** *locale*

<EJ.RangeNavigator
locale='en-US'>

</EJ.RangeNavigator>| **Property:** *locale*

<RangeNavigatorComponent id='charts'
locale='en-US'>
</RangeNavigatorComponent>|

|major grid lines of range navigator| **Property:** *valueAxisSettings.majorGridLines*

var
valueAxisSettings= { majorGridLines: { width: 2, color: 'red' }
};
<EJ.RangeNavigator
locale='en-US'>
</EJ.RangeNavigator>| **Property:**
majorGridLines

<RangeNavigatorComponent id='charts' majorGridLines={ width: 2,
color: 'red'}>
</RangeNavigatorComponent>|

|margin of range navigator| Not Applicable| **Property:** *margin*

<RangeNavigatorComponent
id='charts' margin={ }>
</RangeNavigatorComponent>|

|maximum value of range navigator| **Property:** *valueAxisSettings.range.max*

var
valueAxisSettings= { range: { max: 2 } };
<EJ.RangeNavigator
locale='en-US'>

</EJ.RangeNavigator>| **Property:** *maximum*

<RangeNavigatorComponent
id='charts' maximum={34}>
</RangeNavigatorComponent>|

|minimum value of range navigator| **Property:** *valueAxisSettings.range.min*

var
valueAxisSettings= { range: { min: 2 } };
<EJ.RangeNavigator
locale='en-US'>

</EJ.RangeNavigator>| **Property:** *minimum*

<RangeNavigatorComponent
id='charts' minimum={4}>
</RangeNavigatorComponent>|

|query for data source of range navigator| Not Applicable| **Property:** *query*

<RangeNavigatorComponent id='charts'
query="">
</RangeNavigatorComponent>|

|Secondary label alignment of range navigator| Not Applicable| **Property:** *secondaryLabelAlignment*

<RangeNavigatorComponent id='charts'
secondaryLabelAlignment='Far'>
</RangeNavigatorComponent>|

|Skeleton of range navigator axis| Not Applicable| **Property:** *skeleton*

<RangeNavigatorComponent id='charts'
skeleton="">
</RangeNavigatorComponent>|

|skeletonType of range navigator axis| Not Applicable| **Property:** *skeletonType*

<RangeNavigatorComponent id='charts'
skeletonType="">
</RangeNavigatorComponent>|

|Theme of range navigator| **Property:** *theme*

<EJ.RangeNavigator
theme="">

</EJ.RangeNavigator>| **Property:** *theme*

<RangeNavigatorComponent id='charts'
theme="">
</RangeNavigatorComponent>|

| Default selector value range navigator | **Property:** *selectedRangeSettings*

var selectedRangeSettings= {start:"
end:"};
<EJ.RangeNavigator
selectedRangeSettings={selectedRangeSettings}>

</EJ.RangeNavigator> | **Property:** *value*

<RangeNavigatorComponent id='charts'
value=[2, 10]>
</RangeNavigatorComponent> |

| Value type of range navigator | **Property:** *valueType*

<EJ.RangeNavigator
valueType='DateTime'>
</EJ.RangeNavigator> | **Property:**
valueType

<RangeNavigatorComponent id='charts'
valueType='Logarithmic'>
</RangeNavigatorComponent> |

| Width of range navigator | **Property:** *size.width*

<EJ.RangeNavigator
size={ width:
'200'}>
</EJ.RangeNavigator> | **Property:** *width*

<RangeNavigatorComponent
id='charts' width='400'>
</RangeNavigatorComponent> |

| Height of range navigator | **Property:** *size.height*

<EJ.RangeNavigator
size={ height:
'200'}>
</EJ.RangeNavigator> | **Property:** *height*

<RangeNavigatorComponent
id='charts' height='400'>
</RangeNavigatorComponent> |

| Series settings for range navigator | **Property:** *seriesSettings*

var seriesSettings= {
};
<EJ.RangeNavigator
seriesSettings={seriesSettings}>
</EJ.RangeNavigator> | Not
Applicable |

| Range settings for range navigator | **Property:** *rangeSettings*

var rangeSettings= { start: 3,
end: 6 };
<EJ.RangeNavigator
rangeSettings={rangeSettings}>

</EJ.RangeNavigator> | Not Applicable |

| Scroll range settings for range navigator | **Property:** *scrollRangeSettings*

var
scrollRangeSettings= { start: 3, end: 6
};
<EJ.RangeNavigator
scrollRangeSettings={scrollRangeSettings}>

</EJ.RangeNavigator> | Not Applicable |

Series

<!-- markdownlint-disable MD033 -->

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| animation | **Property:** *enableAnimation*

<EJ.RangeNavigator
enableAnimation={true}>

</EJ.RangeNavigator> | **Property:** *animation.enable*

<RangeNavigatorComponent
id='charts' animation={ enable: true}>
</RangeNavigatorComponent> |

| Border for range navigator series | **Property:** *border*

var series = [{ border: { color:
'transparent', width: 2 } }];
<EJ.RangeNavigator
series={series}>

</EJ.RangeNavigator> | **Property:** *border*

<RangenavigatorSeriesDirective
border={ color: 'pink', width:
2}>
</RangenavigatorSeriesDirective> |

| dataSource for range navigator | **Property:** *series.dataSource*

var series = [{ dataSource: [{}}
];
<EJ.RangeNavigator
series={series}>
</EJ.RangeNavigator> | **Property:**

series.dataSource

```
<br/><br/><RangenavigatorSeriesDirective<br/>dataSource={data}><br/></RangenavigatorSeriesDirective>|
```

| query for data source of range navigator | Not Applicable | **Property:** *query*

```
<br/><br/><RangenavigatorSeriesDirective<br/>dataSource={data}
```

```
query=""><br/></RangenavigatorSeriesDirective>|
```

| series type for range navigator | **Property:** *series.type*

var series = [{ type: "

```
});<br/><EJ.RangeNavigator<br/>series={series}> <br/></EJ.RangeNavigator>| Property: series.type
```

```
<br/><br/><RangenavigatorSeriesDirective<br/>type=""><br/></RangenavigatorSeriesDirective>|
```

| series xName for range navigator | **Property:** *series.xName*

var series = [{ xName: "

```
});<br/><EJ.RangeNavigator<br/>series={series}> <br/></EJ.RangeNavigator>| Property:
```

series.xName

```
<br/><br/><RangenavigatorSeriesDirective<br/>xName=""><br/></RangenavigatorSeriesDirective>|
```

| series yName for range navigator | **Property:** *series.yName*

var series = [{ yName: "

```
});<br/><EJ.RangeNavigator<br/>series={series}> <br/></EJ.RangeNavigator>| Property:
```

series.yName

```
<br/><br/><RangenavigatorSeriesDirective<br/>yName=""><br/></RangenavigatorSeriesDirective>|
```

| series fill color for range navigator | **Property:** *series.fill*

var series = [{ fill: "

```
});<br/><EJ.RangeNavigator<br/>series={series}> <br/></EJ.RangeNavigator>| Property: series.fill
```

```
<br/><br/><RangenavigatorSeriesDirective<br/>fill=""><br/></RangenavigatorSeriesDirective>|
```

| series width for range navigator | **Property:** *series.width*

var series = [{ width: '2'

```
});<br/><EJ.RangeNavigator<br/>series={series}> <br/></EJ.RangeNavigator>| Property:
```

series.width

```
<br/><br/><RangenavigatorSeriesDirective<br/>width=""><br/></RangenavigatorSeriesDirective>|
```

| series dashArray for range navigator | Not Applicable | **Property:** *series.dashArray*

```
<br/><br/><RangenavigatorSeriesDirective<br/>dashArray='10,5'><br/></RangenavigatorSeriesD  
irective>|
```

[StyleSettings](#)

```
<!-- markdownlint-disable MD033 -->
```

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| Style settings of range navigator | **Property:** *navigatorStyleSettings*

var navigatorStyleSettings = { leftThumbTemplate: 'left'

```
};<br/><EJ.RangeNavigator<br/>navigatorStyleSettings={navigatorStyleSettings}>
```

```
<br/></EJ.RangeNavigator>| Property: navigatorStyleSettings
```

```
<br/><br/><RangeNavigatorComponent id='charts'<br/>navigatorStyleSettings={  
unselectedRegionColor: 'transparent' }<br/></RangeNavigatorComponent>|
```

| Selected region color of range navigator | **Property:** *selectedRegionColor*

var navigatorStyleSettings = { selectedRegionColor: 'red' };
<EJ.RangeNavigator
navigatorStyleSettings={navigatorStyleSettings}>
</EJ.RangeNavigator> | **Property:** *selectedRegionColor*

<RangeNavigatorComponent id='charts'
navigatorStyleSettings={selectedRegionColor: 'red'}>
</RangeNavigatorComponent> |

| UnSelected region color of range navigator | **Property:** *unselectedRegionColor*

var navigatorStyleSettings = { unselectedRegionColor: 'red' };
<EJ.RangeNavigator
navigatorStyleSettings={navigatorStyleSettings}>
</EJ.RangeNavigator> | **Property:** *unselectedRegionColor*

<RangeNavigatorComponent id='charts'
navigatorStyleSettings={unselectedRegionColor: 'red'}>
</RangeNavigatorComponent> |

| Thumb color of range navigator | **Property:** *thumbColor*

var navigatorStyleSettings = { thumbColor: 'red' };
<EJ.RangeNavigator
navigatorStyleSettings={navigatorStyleSettings}>
</EJ.RangeNavigator> | **Property:** *thumbSettings*

<RangeNavigatorComponent id='charts'
navigatorStyleSettings={thumbSettings: 'red'}>
</RangeNavigatorComponent> |

| Selected region opacity of range navigator | **Property:** *selectedRegionOpacity*

var navigatorStyleSettings = { selectedRegionOpacity: 0.4 };
<EJ.RangeNavigator
navigatorStyleSettings={navigatorStyleSettings}>
</EJ.RangeNavigator> | Not Applicable |

| Unselected region opacity of range navigator | **Property:** *UnselectedRegionOpacity*

var navigatorStyleSettings = { UnselectedRegionOpacity: 0.4 };
<EJ.RangeNavigator
navigatorStyleSettings={navigatorStyleSettings}>
</EJ.RangeNavigator> | Not Applicable |

| Background for thumb | **Property:** *background*

var navigatorStyleSettings = { background: 'red' };
<EJ.RangeNavigator
navigatorStyleSettings={navigatorStyleSettings}>
</EJ.RangeNavigator> | Not Applicable |

| border for thumb | **Property:** *border*

var navigatorStyleSettings = { border: { color: 'red', width: 2 } };
<EJ.RangeNavigator
navigatorStyleSettings={navigatorStyleSettings}>
</EJ.RangeNavigator> | Not Applicable |

| Highlightsettings for range navigator | **Property:** *highlightSettings*

var navigatorStyleSettings = { highlightSettings: { } };
<EJ.RangeNavigator
navigatorStyleSettings={navigatorStyleSettings}>
</EJ.RangeNavigator> | Not Applicable |

| Selected style settings for range navigator | **Property:** *selectionSettings*

var navigatorStyleSettings = { selectionSettings: { } };
<EJ.RangeNavigator
navigatorStyleSettings={navigatorStyleSettings}>
</EJ.RangeNavigator> | Not Applicable |

| Left thumb template for range navigator | **Property:** *leftThumbTemplate*

var navigatorStyleSettings = { leftThumbTemplate: ''};
<EJ.RangeNavigator
navigatorStyleSettings={navigatorStyleSettings}>
</EJ.RangeNavigator> | Not Applicable |

| Right thumb template for range navigator | **Property:** *rightThumbTemplate*

var navigatorStyleSettings = { rightThumbTemplate: ''};
<EJ.RangeNavigator
navigatorStyleSettings={navigatorStyleSettings}>
</EJ.RangeNavigator> | Not Applicable |

Tooltip

<!-- markdownlint-disable MD033 -->

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| tooltip | **Property:** *visible*

var tooltipSettings = { visible: true};
<EJ.RangeNavigator
tooltipSettings = {tooltipSettings}>
</EJ.RangeNavigator> | **Property:** *enable*

<RangeNavigatorComponent id='charts'>
tooltip= { enable: true }
</RangeNavigatorComponent> |

| background color of tooltip | **Property:** *backgroundColor*

var tooltipSettings = { backgroundColor: 'red'};
<EJ.RangeNavigator
tooltipSettings = {tooltipSettings}>
</EJ.RangeNavigator> | **Property:** *fill*

<RangeNavigatorComponent id='charts'>
tooltip= { fill: 'pink' }
</RangeNavigatorComponent> |

| Font style of tooltip | **Property:** *font*

var tooltipSettings = { font: 'red'};
<EJ.RangeNavigator
tooltipSettings = {tooltipSettings}>
</EJ.RangeNavigator> | **Property:** *textStyle*

<RangeNavigatorComponent id='charts'>
tooltip= { textStyle: 'pink' }
</RangeNavigatorComponent> |

| Label format of tooltip | **Property:** *labelFormat*

var tooltipSettings = {labelFormat: 'yMd'};
<EJ.RangeNavigator
tooltipSettings = {tooltipSettings}>
</EJ.RangeNavigator> | **Property:** *format*

<RangeNavigatorComponent id='charts'>
tooltip= { format: 'yMd' }
</RangeNavigatorComponent> |

| Display mode of tooltip | **Property:** *tooltipDisplayMode*

var tooltipSettings = { tooltipDisplayMode: 'always'};
<EJ.RangeNavigator
tooltipSettings = {tooltipSettings}>
</EJ.RangeNavigator> | **Property:** *displayMode*

<RangeNavigatorComponent id='charts'>
tooltip= {displayMode: 'Always'}
</RangeNavigatorComponent> |

| Template of tooltip | Not Applicable | **Property:** *template*

<RangeNavigatorComponent id='charts'>
tooltip= {template: '<div>Chart</div>'}
</RangeNavigatorComponent> |

| Border of tooltip | Not Applicable | **Property:** *border*

<RangeNavigatorComponent id='charts'>
tooltip= {border: { color: 'red', width: 2 }}
</RangeNavigatorComponent> |

| Border of tooltip | Not Applicable | **Property:** *opacity*

<RangeNavigatorComponent id='charts'>
tooltip= {opacity: 0.5 }
</RangeNavigatorComponent> |

Period Selector

```
<!-- markdownlint-disable MD033 -->
```

```
| Behavior | API in Essential JS 1 | API in Essential JS 2 |
```

```
| --- | --- | --- |
```

```
| period Selector position | Not Applicable | Property: periodSelectorSettings.position
```

```
<br/><br/><RangeNavigatorComponent id='charts'><br/>periodSelectorSettings= {  
position:'Top'}<br/></RangeNavigatorComponent>|
```

Methods

```
<!-- markdownlint-disable MD033 -->
```

```
| Behavior | API in Essential JS 1 | API in Essential JS 2 |
```

```
| --- | --- | --- |
```

```
| Print | Not Applicable | Property: print() <br/><br/>public
```

```
clickHandler(){<br/>this.range.print();}<br/><RangeNavigatorComponent id='charts'><br/>ref={g =>  
this.range = g}<br/></RangeNavigatorComponent>|
```

```
| Export | Not Applicable | Property: export() <br/><br/>public
```

```
clickHandler(){<br/>this.range.export('PNG','export');}<br/><RangeNavigatorComponent  
id='charts'><br/>ref={g => this.range = g}<br/></RangeNavigatorComponent>|
```

Events

```
<!-- markdownlint-disable MD033 -->
```

```
| Behavior | API in Essential JS 1 | API in Essential JS 2 |
```

```
| --- | --- | --- |
```

```
| Fires before loading the RangeNavigator. | Property: load <br/><br/><EJ.RangeNavigator<br/>load =  
{Load}> <br/></EJ.RangeNavigator><br/>function Load(){ };| Property: load  
<br/><br/><RangeNavigatorComponent  
id='charts'><br/>load={this.load.bind(this)}<br/></RangeNavigatorComponent><br/> public load():  
void { };|
```

```
| Fires before loading the RangeNavigator. | Property: loaded
```

```
<br/><br/><EJ.RangeNavigator<br/>loaded = {loaded}> <br/></EJ.RangeNavigator><br/>function  
Loaded(){ };| Property: loaded <br/><br/><RangeNavigatorComponent  
id='charts'><br/>loaded={this.loaded.bind(this)}<br/></RangeNavigatorComponent><br/> public  
loaded(): void { };|
```

```
| Fires when the value changes in range navigator | Property: rangeChanged
```

```
<br/><br/><EJ.RangeNavigator<br/>rangeChanged = {rangeChanged}>  
<br/></EJ.RangeNavigator><br/>function rangeChanged(){ };| Property: changed  
<br/><br/><RangeNavigatorComponent  
id='charts'><br/>changed={this.changed.bind(this)}<br/></RangeNavigatorComponent><br/>  
public changed(): void { };|
```

```
| Fires after the RangeNavigator | Not Applicable | Property: resized
```

```
<br/><br/><RangeNavigatorComponent
```

```
id='charts'<br/>resized={this.resized.bind(this)}<br/></RangeNavigatorComponent><br/> public
resized(): void { };|
```

| Fires before tooltip in RangeNavigator | Not Applicable | **Property:** *tooltipRender*

```
<br/><br/><RangeNavigatorComponent
id='charts'<br/>tooltipRender={this.tooltipRender.bind(this)}<br/></RangeNavigatorComponent
><br/> public tooltipRender(): void { };|
```

| Fires before period render in the RangeNavigator | Not Applicable | **Property:** *selectorRender*

```
<br/><br/><RangeNavigatorComponent
id='charts'<br/>selectorRender={this.selectorRender.bind(this)}<br/></RangeNavigatorComponen
t><br/> public selectorRender(): void { };|
```

| Fires when scrollStart the RangeNavigator | **Property:** *scrollStart*

```
<br/><br/><EJ.RangeNavigator<br/>scrollStart = {scrollStart}>
<br/></EJ.RangeNavigator><br/>function scrollStart(){ };| Not Applicable |
```

| Fires when scrollEnd the RangeNavigator | **Property:** *scrollEnd*

```
<br/><br/><EJ.RangeNavigator<br/>scrollEnd = {scrollEnd}>
<br/></EJ.RangeNavigator><br/>function scrollEnd(){ };| Not Applicable |
```

| Fires when selected range Start the RangeNavigator | **Property:** *selectedRangeStart*

```
<br/><br/><EJ.RangeNavigator<br/>selectedRangeStart = {selectedRangeStart}>
<br/></EJ.RangeNavigator><br/>function selectedRangeStart(){ };| Not Applicable |
```

| Fires when selected range ends the RangeNavigator | **Property:** *selectedRangeEnd*

```
<br/><br/><EJ.RangeNavigator<br/>selectedRangeEnd = {selectedRangeEnd}>
<br/></EJ.RangeNavigator><br/>function selectedRangeEnd(){ };| Not Applicable |
```

| Fires when scroll range changed in the RangeNavigator | **Property:** *scrollChanged*

```
<br/><br/><EJ.RangeNavigator<br/>scrollChanged =
{scrollChanged}><br/></EJ.RangeNavigator><br/>function scrollChanged(){ };| Not Applicable |
```

| Fires when click in the RangeNavigator | **Property:** *click*

<EJ.RangeNavigator
click =
{click}>
</EJ.RangeNavigator>
function click(){ };| Not Applicable |

| Fires when right click in the RangeNavigator | **Property:** *rightClick*

```
<br/><br/><EJ.RangeNavigator<br/>rightClick =
{rightClick}><br/></EJ.RangeNavigator><br/>function rightClick(){ };| Not Applicable |
```

| Fires when double click in the RangeNavigator | **Property:** [Link to the Video](#)

```
<br/><br/><EJ.RangeNavigator<br/>doubleClick =
{doubleClick}><br/></EJ.RangeNavigator><br/>function doubleClick(){ };| Not Applicable |
```

Range Slider

Getting Started

The following section explains the required steps to build the simple Slider component with its basic usage in step by step procedure.

To get start quickly with React Range Slider, you can check on this video:

Dependencies

Install the below required dependent packages to render the Slider component.

```
`javascript
|-- @syncfusion/ej2-react-inputs
|-- @syncfusion/ej2-react-base
|-- @syncfusion/ej2-react-popups
|-- @syncfusion/ej2-react-buttons
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
`,`
```

Installation and Configuration

You can use [create-react-app](#) to setup the applications.

To install `create-react-app` run the following command.

```
`bash
npm install -g create-react-app
`,`
```

To set-up a React application in TypeScript environment, run the following command.

```
`
npx create-react-app my-app --template typescript
cd my-app
npm start
`,`
```

To set-up a React application in JavaScript environment, run the following command.

```
`
npx create-react-app my-app
cd my-app
npm start
`,`
```

Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) public registry. Now, we are going to render `Slider` component from these packages.

To install **Slider** component, use the following command.

```
`bash
npm install @syncfusion/ej2-react-inputs --save
`
```

The above command installs [Slider dependencies](#) which are required to render the component in the **React** environment.

Adding CSS Reference

Import **Slider** component required theme references at the top of **src/App.css**.

```
`css
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-react-inputs/styles/material.css";
`
```

We can also use [CRG](#) to generate combined component styles.

Adding Slider component

Now, you can add **Slider** component in the application. For getting started, add **Slider** component in **src/App.tsx** file using the following code snippet.

```
`ts
import * as React from 'react';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
import './App.css';
function App() {
  return (
    <div id='container'>
      <div className='wrap'>
        <SliderComponent id='slider' value={30} />
      </div>
    </div>
  );
}
export default App;
`
```

Run the Application

The Essential JS 2 quickstart application project is configured to compile and run the application in browser. Use the following command to run the application.

`

npm start

`

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    return (
        <div id='container'>
            <div className='wrap'>
                <div className='sliderwrap'>
                    <SliderComponent id='default' value={30}/>
                </div>
            </div>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    return (
        <div id='container'>
            <div className='wrap'>
                <div className='sliderwrap'>
                    <SliderComponent id='default' value={30} />
                </div>
            </div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Types

The types of Slider are as follows:

| **Types** | **Usage** |

| --- | --- |

| Default | Shows a default Slider to select a single value. |

| MinRange | Displays the shadow from the start value to the current selected value. |

| Range | Selects a range of values. It also displays the shadow in-between the selection range. |

Both the Default Slider and Min-Range Slider have same behavior that is used to select a single value.

In Min-Range Slider, a shadow is considered from the start value to current handle position. But the Range Slider

contains two handles that is used to select a range of values and a shadow is considered in between the two handles.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    return (
        <div id='container'>
            <div className='wrap'>
                <div className="sliderwrap">
                    <div className="labeltext">Default</div>
                    <SliderComponent id='default' value={30}/>
                </div>
                <div className="sliderwrap">
                    <div className="labeltext">MinRange</div>
                    <SliderComponent id='minrange' type='MinRange'
value={30}/>
                </div>
                <div className="sliderwrap">
                    <div className="labeltext">Range</div>
                    <SliderComponent id='range' type='Range' value={[30,
70]}/>
                </div>
            </div>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    return (
        <div id='container'>
            <div className='wrap'>
                <div className="sliderwrap">
                    <div className="labeltext">Default</div>
                    <SliderComponent id='default' value={30}/>
                </div>
                <div className="sliderwrap">
                    <div className="labeltext">MinRange</div>
                    <SliderComponent id='minrange' type='MinRange' value={30}
/>
                </div>
                <div className="sliderwrap">
                    <div className="labeltext">Range</div>
                    <SliderComponent id='range' type='Range' value={[30,
70]}/>
                </div>
            </div>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

```

        <div className="labeltext">Range</div>
        <SliderComponent id='range' type='Range' value={[30, 70]}
    />
    </div>
</div>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Customization

Orientation

The Slider can be displayed, either in horizontal or vertical orientation. By default, the Slider renders in horizontal orientation.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    return (<div id='container'>
        <div className='wrap'>
            <SliderComponent id='slider' orientation='Vertical'
value={30}/>
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    return (
        <div id='container'>
            <div className='wrap'>
                <SliderComponent id='slider' orientation='Vertical'
value={30} />
            </div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Tooltip

The Slider displays the tooltip to indicate the current value by clicking the Slider bar or drag the Slider handle. The Tooltip position can be customized by using the `placement` property.

Also decides the tooltip display mode on a page, i.e., on hovering, focusing, or clicking on the Slider handle and it always remains/displays on the page.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let tooltip = { placement: 'After', isVisible: true, showOn: 'Always' };
    return (<div id='container'>
        <div className='wrap'>
            <SliderComponent id='slider' type="MinRange"
tooltip={tooltip} value={30}/>
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let tooltip: Object = { placement: 'After', isVisible: true, showOn:
'Always' };
    return (
        <div id='container'>
            <div className='wrap'>
                <SliderComponent id='slider' type="MinRange"
tooltip={tooltip} value={30} />
            </div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Buttons

The Slider value can be changed by using the Increase and Decrease buttons. In Range Slider, by default the first handle value will be changed while clicking the button. Change the handle focus and press the button to change the last focused handle value.

After enabling the slider buttons if the 'Tab' key is pressed, the focus goes to the handle and not to the button.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    return (<div id='container'>
```

```

        <div className='wrap'>
            <SliderComponent id='slider' showButtons={true} type='Range'
value={ [30, 70]} />
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    return (
        <div id='container'>
            <div className='wrap'>
                <SliderComponent id='slider' showButtons={true} type='Range'
value={ [30, 70]} />
            </div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

See Also

[Slider Formatting](#)

[Ticks in Slider](#)

[Limits in Slider](#)

Ticks in React Range slider component

The Ticks in Slider supports you to easily identify the current value/values of the Slider. It contains `smallStep` and `largeStep`. The value of the major ticks alone will be displayed in the slider. In order to enable/disable the small ticks, use the `showSmallTicks` property.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let tooltip = { placement: "Before", isVisible: true, showOn: "Always" };
    let value = 30;
    // Slider ticks customization
    let ticks = {
        placement: "After",
        largeStep: 20,
        smallStep: 10,
        showSmallTicks: true
    };
    return (<div id="container">

```

```

        <div className="wrap">
            <SliderComponent id="slider" value={value} tooltip={tooltip}
ticks={ticks}/>
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let tooltip: TooltipDataModel = { placement: "Before", isVisible: true,
showOn: "Always" };
    let value = 30;
    // Slider ticks customization
    let ticks: TicksDataModel = {
        placement: "After",
        largeStep: 20,
        smallStep: 10,
        showSmallTicks: true
    };
    return (
        <div id="container">
            <div className="wrap">
                <SliderComponent id="slider" value={value} tooltip={tooltip}
ticks={ticks} />
            </div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Step

When the Slider is moved, it increases/decreases the value based on the step value. By default, the value is increased/decreased by 1. Use the `step` property to change the increment step value.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let ticks = {
        placement: "After",
        largeStep: 20,
        smallStep: 10,
        showSmallTicks: true
    };
    let tooltip = { placement: "Before", isVisible: true, showOn: "Always" };
    let value = 30;

```

```
// Enables step
let step = 10;
return (<div id="container">
  <div className="wrap">
    <SliderComponent id="slider" value={value} step={step}
      tooltip={tooltip} ticks={ticks}/>
  </div>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let ticks: TicksDataModel = {
    placement: "After",
    largeStep: 20,
    smallStep: 10,
    showSmallTicks: true
  };
  let tooltip: TooltipDataModel = { placement: "Before", isVisible: true,
    showOn: "Always" };
  let value = 30;
  // Enables step
  let step = 10;
  return (
    <div id="container">
      <div className="wrap">
        <SliderComponent
          id="slider"
          value={value}
          step={step}
          tooltip={tooltip}
          ticks={ticks}
        />
      </div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Min and Max

Enables the minimum/starting and maximum/ending value of the Slider, by using the **min** and **max** property. By default, the minimum value is 1 and maximum value is 100. In the following sample the slider is rendered with the min value as 100 and max value as 1000.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
```



```
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let ticks = {
    placement: "After",
    largeStep: 200,
    smallStep: 100,
    showSmallTicks: true
  };
  let tooltip = { placement: "Before", isVisible: true, showOn: "Always" };
  // Minimum value
  let min = 100;
  // Maximum value
  let max = 1100;
  // Slider current value
  let value = 400;
  return (<div id="container">
    <div className="wrap">
      <SliderComponent id="slider" min={min} max={max} value={value}
tooltip={tooltip} ticks={ticks}/>
    </div>
  </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let ticks: TicksDataModel = {
    placement: "After",
    largeStep: 200,
    smallStep: 100,
    showSmallTicks: true
  };
  let tooltip: TooltipDataModel = { placement: "Before", isVisible: true,
showOn: "Always" };
  // Minimum value
  let min = 100;
  // Maximum value
  let max = 1100;
  // Slider current value
  let value = 400;
  return (
    <div id="container">
      <div className="wrap">
        <SliderComponent
          id="slider"
          min={min}
          max={max}
          value={value}
          tooltip={tooltip}
          ticks={ticks}
        />
      </div>
    </div>
  );
}
```

```

    </div>
  </div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Format in React Range slider component

The **format** feature used to customize the units of Slider values to desired format. The formatted values will also be applied to the ARIA attributes of the slider. There are two ways of achieving formatting in slider.

- Use the **format** API of slider which utilizes our **Internationalization** to format values.
- Customize using the events namely **renderingTicks** and **tooltipChange**.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let tooltip = { isVisible: true, format: "C2" };
  let value = 30;
  // Slider ticks customization
  let ticks = {
    placement: "After",
    format: "C2",
    largeStep: 20,
    smallStep: 10,
    showSmallTicks: true
  };
  return (
    <div id="container">
      <div className="wrap">
        <SliderComponent id="slider" min={0} max={100} value={value}
          tooltip={tooltip} ticks={ticks}/>
      </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let tooltip = { isVisible: true, format: "C2" };
  let value = 30;
  // Slider ticks customization
  let ticks: TicksDataModel = {
    placement: "After",
    format: "C2",

```

```

    largeStep: 20,
    smallStep: 10,
    showSmallTicks: true
  };
  return (
    <div id="container">
      <div className="wrap">
        <SliderComponent
          id="slider"
          min={0}
          max={100}
          value={value}
          tooltip={tooltip}
          ticks={ticks}
        />
      </div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Using format API

In this method, we have different predefined formatting styles like Numeric (N), Percentage (P), Currency (C) and # specifiers. In this below example we have formatted the ticks and tooltip values into percentage.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let tooltip = {
    placement: "Before",
    isVisible: true,
    showOn: "Always",
    format: "P0"
  };
  let value = 0.3;
  // Slider ticks customization
  let ticks = {
    placement: "After",
    largeStep: 0.2,
    smallStep: 0.1,
    showSmallTicks: true,
    format: "P0"
  };
  return (<div id="container">
    <div className="wrap">
      <SliderComponent id="slider" min={0} max={1} step={0.01}
value={value} tooltip={tooltip} ticks={ticks}/>
    </div>
  </div>);
}
export default App;

```

```
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let tooltip: TooltipDataModel = {
    placement: "Before",
    isVisible: true,
    showOn: "Always",
    format: "P0"
  };
  let value = 0.3;
  // Slider ticks customization
  let ticks: TicksDataModel = {
    placement: "After",
    largeStep: 0.2,
    smallStep: 0.1,
    showSmallTicks: true,
    format: "P0"
  };
  return (
    <div id="container">
      <div className="wrap">
        <SliderComponent
          id="slider"
          min={0}
          max={1}
          step={0.01}
          value={value}
          tooltip={tooltip}
          ticks={ticks}
        />
      </div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Using Events

In this method, we will be retrieving the values from the slider events then process them to desired formatted the values.

In this sample we have customized the **ticks** values into weekdays as one formatting and **tooltip** values into day of the week as another formatting.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
```

```

let tooltip = { placement: "Before", isVisible: true };
let value = 2;
// Slider ticks customization
let ticks = { placement: "After", largeStep: 1 };
function renderingTicksHandler(args) {
    // Weekdays Array
    let daysArr = [
        "Sunday",
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday"
    ];
    // Customizing each ticks text into weekdays
    args.text = daysArr[parseFloat(args.value.toString())];
}
function tooltipChangeHandler(args) {
    // Customizing tooltip to display the Day (in numeric) of the week
    args.text = "Day " + (Number(args.value) + 1).toString();
}
return (<div id="container">
    <div className="wrap">
        <SliderComponent id="slider" min={0} max={6} step={1} value={value}
        tooltip={tooltip} ticks={ticks}
        tooltipChange={tooltipChangeHandler.bind(this)}
        renderingTicks={renderingTicksHandler.bind(this)} />
    </div>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let tooltip: TooltipDataModel = { placement: "Before", isVisible: true };
    let value = 2;
    // Slider ticks customization
    let ticks: TicksDataModel = { placement: "After", largeStep: 1 };
    function renderingTicksHandler(args: SliderTickEventArgs) {
        // Weekdays Array
        let daysArr: string[] = [
            "Sunday",
            "Monday",
            "Tuesday",
            "Wednesday",
            "Thursday",
            "Friday",
            "Saturday"
        ];
    };
    // Customizing each ticks text into weekdays

```

```

    args.text = daysArr[parseFloat(args.value.toString())];
  }
  function tooltipChangeHandler(args: SliderTooltipEventArgs) {
    // Customizing tooltip to display the Day (in numeric) of the week
    args.text = "Day " + (Number(args.value) + 1).toString();
  }
  return (
    <div id="container">
      <div className="wrap">
        <SliderComponent
          id="slider"
          min={0}
          max={6}
          step={1}
          value={value}
          tooltip={tooltip}
          ticks={ticks}
          tooltipChange={tooltipChangeHandler.bind(this) as any}
          renderingTicks={renderingTicksHandler.bind(this) as any}
        />
      </div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Limits in React Range slider component

The slider limits restrict the slider thumb between a particular range. This is used if higher or lower value affects the process or product where the slider is being used.

The following are the six options in the slider's limits object. Each API in the limits object is optional.

- enabled: Enables the limits in the Slider.
- minStart: Sets the minimum limit for the first handle.
- minEnd: Sets the maximum limit for the first handle.
- maxStart: Sets the minimum limit for the second handle.
- maxEnd: Sets the maximum limit for the second handle.
- startHandleFixed: Locks the first handle.
- endHandleFixed: Locks the second handle.

Default and MinRange Slider limits

There is only one handle in the Default and MinRange Slider, so minStart, minEnd, and startHandleFixed options can be used.

When the limits are enabled in the Slider, the limited area becomes darken. So you can differentiate the allowed and restricted area.

Refer to the following snippet to enable the limits in the Slider.

```

`ts
.....

```

```
limits: { enabled: true, minStart: 10, minEnd: 40 }
```

```
.....
```

```
,
```

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let limits = { enabled: true, minStart: 10, minEnd: 40 };
  let tooltip = { isVisible: true };
  let value = 30;
  return (<div id='container'>
    <div className='wrap'>
      <div className="sliderwrap">
        <SliderComponent value={value} type='MinRange' min={0}
max={100} limits={limits} tooltip={tooltip}/>
      </div>
    </div>
  </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let limits: object = { enabled: true, minStart: 10, minEnd: 40 };
  let tooltip: object = { isVisible: true };
  let value: number = 30;
  return (
    <div id='container'>
      <div className='wrap'>
        <div className="sliderwrap">
          <SliderComponent value={value} type='MinRange' min={0}
max={100} limits={limits}
          tooltip={tooltip} />
        </div>
      </div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Range Slider limits

In the range slider, both handles can be restricted and locked from the limit's object. In this sample, the first handle is limited between 10 and 40, and the second handle is limited between 60 and 90.

```
`ts
```

.....

limits: { enabled: true, minStart: 10, minEnd: 40, maxStart: 60, maxEnd: 90 }

.....

,

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let limits = { enabled: true, minStart: 10, minEnd: 40, maxStart: 60,
maxEnd: 90 };
    let tooltip = { isVisible: true };
    let value = [30, 70];
    return (<div id='container'>
        <div className='wrap'>
            <div className="sliderwrap">
                <SliderComponent value={value} type='Range' min={0}
max={100} limits={limits} tooltip={tooltip}/>
            </div>
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let limits: object = { enabled: true, minStart: 10, minEnd: 40, maxStart:
60, maxEnd: 90 };
    let tooltip: object = { isVisible: true };
    let value: number[] = [30, 70];
    return (
        <div id='container'>
            <div className='wrap'>
                <div className="sliderwrap">
                    <SliderComponent value={value} type='Range' min={0}
max={100} limits={limits}
                    tooltip={tooltip} />
                </div>
            </div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```


Handle lock

The movement of slider handles can be locked by enabling the `startHandleFixed` and `endHandleFixed` properties in the limit's object.

In this sample, the movement of both slider handles has been locked.

```
`ts
```

```
.....
```

```
limits: { enabled: true, startHandleFixed: true, endHandleFixed: true }
```

```
.....
```

```
`
```

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let limits = { enabled: true, startHandleFixed: true, endHandleFixed:
true };
    let tooltip = { isVisible: true };
    let value = [30, 70];
    return (<div id='container'>
        <div className='wrap'>
            <div className="sliderwrap">
                <SliderComponent value={value} type='Range' min={0}
max={100} limits={limits} tooltip={tooltip}/>
            </div>
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import { Slider } from '@syncfusion/ej2-inputs';
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let limits: object = { enabled: true, startHandleFixed: true,
endHandleFixed: true };
    let tooltip: object = { isVisible: true };
    let value: number[] = [30, 70];
    return (
        <div id='container'>
            <div className='wrap'>
                <div className="sliderwrap">
                    <SliderComponent value={value} type='Range' min={0}
max={100} limits={limits}
                    tooltip={tooltip} />
                </div>
            </div>
        </div>
    );
}
```

```

    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Style in React Range slider component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

Customizing the slider track

Use the following CSS to customize the slider track.

```

`css

.e-control-wrapper.e-slider-container.e-horizontal .e-slider-track {
background: #007bff;
height: 3px;
}
`

```

Customizing the slider handle

Use the following CSS to customize the slider handle properties.

```

`css

.e-control-wrapper.e-slider-container .e-slider .e-handle {
background-color: #f9920b;
border-radius: 50%;
border: 0;
}
`

```

Customizing the slider limits

Use the following CSS to customize the slider limits.

```

`css

.e-control-wrapper.e-slider-container.e-horizontal .e-limits {
background-color: rgba(69, 100, 233, 0.46);
}
`

```

Customizing the slider ticks

Use the following CSS to customize the slider ticks.

```

`css

.e-scale .e-tick.e-custom::before {

```

```
content: '\e967';
position: absolute;
}
`
```

Customizing the slider buttons

Use the following CSS to customize the slider buttons.

```
`css
.e-control-wrapper.e-slider-container .e-slider-button {
background: #007bff;
height: 25px;
width: 25px;
}
`
```

Accessibility in React Range Slider component

The Range Slider component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Range Slider component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

| Screen Reader Support | |

| Right-To-Left Support | |

| Color Contrast | |

| Mobile Device Support | |

| Keyboard Navigation Support | |

| [Accessibility Checker](#) Validation | |

| [Axe-core](#) Accessibility Validation | |

```

<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

```

WAI-ARIA attributes

The Range Slider component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Range Slider component:

Attributes	Purpose
---	---
<code>role=slider</code>	Used to convey a significant and contextual message to the user.
<code>aria-valuemin</code>	Indicates the Minimum value of the slider.
<code>aria-valuemax</code>	Indicates the Maximum value of the slider.
<code>aria-valuenow</code>	Indicates the current value of the slider.
<code>aria-valuetext</code>	Returns the current text of the slider.
<code>aria-orientation</code>	Indicates whether the Slider is oriented horizontally or vertically.
<code>aria-label</code>	Provides an accessible name for the Slider, serving as label text for the Slider's left and right buttons (for increment and decrement).

Keyboard interaction

The Range Slider component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Range Slider component.

Press	To do this
---	---
Right Arrow/Up Arrow	Increase the Slider value.
Left Arrow/Down Arrow	Decrease the Slider value.
Home	Moves to the start value (for Range Slider when the second thumb is focused and the Home key is pressed, it moves to the first thumb value).

| `<End>` | Moves to the end value (for Range Slider when the first thumb is focused and the End key is pressed, it moves to the second thumb value). |

| `Page Up` | Increases the Slider by `largeStep` value. |

| `Page Down` | Decreases the Slider by `largeStep` value. |

Ensuring accessibility

The Range Slider component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Range Slider component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Range Slider component with accessibility tools.

See also

- [Accessibility in Syncfusion React components](#)

Ej1 api migration in React Range slider component

This article describes the API migration process of Slider component from Essential JS 1 to Essential JS 2

{% raw %}

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| Max value | **Property:** `maxValue` `
` `<EJ.Slider maxValue={60} />` | **Property:** `max` `
` `<SliderComponent id='slider' max={100} />` |

| Min value | **Property:** `minValue` `
` `<EJ.Slider minValue={60} />` | **Property:** `min` `
` `<SliderComponent id='slider' min={100} />` |

| Step | **Property:** `incrementStep`, `largeStep`, `smallStep`, `showSmallTicks` `
` `<EJ.Slider incrementStep={20} smallStep={5} largeStep={40} showSmallTicks={true} />` | **Property:** `ticks` `
` `public ticks = { placement: 'After', largeStep: 20, smallStep: 10, showSmallTicks: true };
` `<SliderComponent id='slider' ticks={this.ticks} />` |

| Type | **Property:** `sliderType` `
` `<SliderComponent id='slider' value={30} sliderType="minrange" />` | **Property:** `type` `
` `<SliderComponent id='slider' type='Range' />` |

| Tooltip | **Property:** `showTooltip` `
` `<SliderComponent id='slider' value={30} showTooltip={true} />` | **Property:** `tooltip` `
` `public tooltip = { placement: 'Before', isVisible: true, showOn: 'Always' };
` `<SliderComponent id='slider' tooltip={this.tooltip} />` |

| RTL | **Property:** `enableRTL` `
` `<SliderComponent id='slider' value={30} enableRTL={true} />` | **Property:** `enableRtl` `
` `<SliderComponent id='slider' enableRtl={false} />` |

| Custom values | **Not Applicable** | **Property:** `customValues` `
` `public customValues = ['Mon', 'Tue', 'Wed'];
` `<SliderComponent id='slider' customValues={this.customValues} />` |

| Limit the slider movement | **Not Applicable** | **Property:** *limits*
 public limits = { enabled: true, minStart: 10, minEnd: 40 };
 <SliderComponent id='slider' type='MinRange' limits={this.limits} />|

| Disable | **Method:** *disable*
 <SliderComponent id='slider' value={30} />
 public sliderObj = \${"#slider"}.ejSlider("instance"); sliderObj.disable(); | **Property:** *enabled*
 <SliderComponent id='slider' enable={false} value={50} /> |

| Enable | **Method:** *enable*
 <SliderComponent id='slider' value={30} />
 public sliderObj = \${"#slider"}.ejSlider("instance"); sliderObj.enable(); | **Property:** *enabled*
 <SliderComponent id='slider' enable={true} value={50} /> |

| Set Value | **Method:** *setValue(value,[enableAnimation])*
 <SliderComponent id='slider' value={30} />
 public sliderObj = \${"#slider"}.ejSlider("instance"); sliderObj.setValue(50); | **Property:** *value*
 <SliderComponent id='slider' value={this.state.value} />
 this.setState({value: 100}); |

| Get Value | **Method:** *getValue()*
 <SliderComponent id='slider' value={30} />
 public sliderObj = \${"#slider"}.ejSlider("instance"); sliderObj.getValue(); | **Property:** *value*
 <SliderComponent id='slider' value={30} />
 public sliderValue = this.sliderObj.value; |

| Destroy | **Not Applicable** | **Method:** *destroy()*
 <SliderComponent id='slider' value={30} />
 public sliderValue = this.sliderObj.destroy(); |

| Change | **Event:** *change*
 public change(args) { }
 <SliderComponent id='slider' value={30} change={this.change} /> | **Event:** *changed*
 public changed(args): void { } ;
 <SliderComponent id='slider' value={30} changed={this.changed} /> |

| Create | **Event:** *create*
 public create(args) { }
 <SliderComponent id='slider' value={30} create={this.create} /> | **Event:** *created*
 public created(args): void { } ;
 <SliderComponent id='slider' value={30} created={this.created} /> |

| Slide | **Event:** *slide*
 public slide(args) { }
 <SliderComponent id='slider' value={30} slide={this.slide} /> | **Event:** *change*
 public change(args): void { } ;
 <SliderComponent id='slider' value={30} change={this.change} /> |

| Start | **Event:** *start*
 public start(args) { }
 <SliderComponent id='slider' value={30} start={this.start} /> | **Event:** *created*
 public created(args): void { } ;
 <SliderComponent id='slider' value={30} created={this.created} /> |

| Stop | **Event:** *stop*
 public stop(args) { }
 <SliderComponent id='slider' value={30} stop={this.stop} /> | **Event:** *changed*
 public changed(args): void { } ;
 <SliderComponent id='slider' value={30} changed={this.changed} /> |

| Rendered Ticks | **Not Applicable** | **Event:** *renderedTicks*
 public renderedTicks(args): void { } ;
 <SliderComponent id='slider' value={30} renderedTicks={this.renderedTicks} /> |

{% endraw %}

How To

Time range slider in React Range slider component

The time formatting can be achieved same as the date formatting using `renderingTicks` and `change` events. The process of time formatting is explained in the below sample.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let min = new Date(2013, 6, 13, 11).getTime();
    let max = new Date(2013, 6, 13, 17).getTime();
    let value = new Date(2013, 6, 13, 13).getTime();
    let step = 3600000;
    // Slider ticks customization
    let ticks = { placement: "After", largeStep: 2 * 3600000 };
    let tooltip = { placement: "Before", isVisible: true };
    function renderingTicksHandler(args) {
        let totalMiliSeconds = Number(args.value);
        let custom = { hour: "2-digit", minute: "2-digit" };
        args.text = new Date(totalMiliSeconds).toLocaleTimeString("en-us",
custom);
    }
    function tooltipChangeHandler(args) {
        let totalMiliSeconds = Number(args.text);
        let custom = { hour: "2-digit", minute: "2-digit" };
        args.text = new Date(totalMiliSeconds).toLocaleTimeString("en-us",
custom);
    }
    return (<div id="container">
        <div className="wrap">
            <SliderComponent id="slider" min={min} max={max} value={value}
step={step} tooltip={tooltip} ticks={ticks} showButtons={true}
tooltipChange={tooltipChangeHandler.bind(this)}
renderingTicks={renderingTicksHandler.bind(this)} />
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let min = new Date(2013, 6, 13, 11).getTime();
    let max = new Date(2013, 6, 13, 17).getTime();
    let value = new Date(2013, 6, 13, 13).getTime();
    let step = 3600000;
    // Slider ticks customization
    let ticks: TicksDataModel = { placement: "After", largeStep: 2 * 3600000 };
    let tooltip: TooltipDataModel = { placement: "Before", isVisible: true };
    function renderingTicksHandler(args: SliderTickEventArgs) {
```

```

    let totalMiliSeconds = Number(args.value);
    let custom = { hour: "2-digit", minute: "2-digit" };
    args.text = new Date(totalMiliSeconds).toLocaleTimeString("en-us",
custom);
  }
  function tooltipChangeHandler(args: SliderTooltipEventArgs) {
    let totalMiliSeconds = Number(args.text);
    let custom = { hour: "2-digit", minute: "2-digit" };
    args.text = new Date(totalMiliSeconds).toLocaleTimeString("en-us",
custom);
  }
  return (
    <div id="container">
      <div className="wrap">
        <SliderComponent
          id="slider"
          min={min}
          max={max}
          value={value}
          step={step}
          tooltip={tooltip}
          ticks={ticks}
          showButtons={true}
          tooltipChange={tooltipChangeHandler.bind(this) as any}
          renderingTicks={renderingTicksHandler.bind(this) as any}
        />
      </div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Date range slider in React Range slider component

The Date formatting can be achieved in `ticks` and `tooltip` using `renderingTicks` and `tooltipChange` events respectively. The process of formatting is explained in the below sample.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let min = new Date("2013-06-13").getTime();
  let max = new Date("2013-06-21").getTime();
  let step = 86400000;
  let value = new Date("2013-06-15").getTime();
  // Slider ticks customization
  let ticks = { placement: "After", largeStep: 2 * 86400000 };
  let tooltip = { placement: "Before", isVisible: true };
  function renderingTicksHandler(args) {
    let totalMiliSeconds = Number(args.value);
    // Converting the current milliseconds to the respective date in
    desired format
    let custom = { year: "numeric", month: "short", day: "numeric" };

```



```

        args.text = new Date(totalMiliSeconds).toLocaleDateString("en-us",
custom);
    }
    function tooltipChangeHandler(args) {
        let totalMiliSeconds = Number(args.text);
        // Converting the current milliseconds to the respective date in
desired format
        let custom = { year: "numeric", month: "short", day: "numeric" };
        args.text = new Date(totalMiliSeconds).toLocaleDateString("en-us",
custom);
    }
    return (<div id="container">
        <div className="wrap">
            <SliderComponent id="slider" min={min} max={max} value={value}
step={step} tooltip={tooltip} ticks={ticks} showButtons={true}
tooltipChange={tooltipChangeHandler.bind(this)}
renderingTicks={renderingTicksHandler.bind(this)} />
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let min = new Date("2013-06-13").getTime();
    let max = new Date("2013-06-21").getTime();
    let step = 86400000;
    let value = new Date("2013-06-15").getTime();
    // Slider ticks customization
    let ticks: TicksDataModel = { placement: "After", largeStep: 2 * 86400000
};
    let tooltip: TooltipDataModel = { placement: "Before", isVisible: true };
    function renderingTicksHandler(args: SliderTickEventArgs) {
        let totalMiliSeconds = Number(args.value);
        // Converting the current milliseconds to the respective date in desired
format
        let custom = { year: "numeric", month: "short", day: "numeric" };
        args.text = new Date(totalMiliSeconds).toLocaleDateString("en-us",
custom);
    }
    function tooltipChangeHandler(args: SliderTooltipEventArgs) {
        let totalMiliSeconds = Number(args.text);
        // Converting the current milliseconds to the respective date in desired
format
        let custom = { year: "numeric", month: "short", day: "numeric" };
        args.text = new Date(totalMiliSeconds).toLocaleDateString("en-us",
custom);
    }
    return (
        <div id="container">
            <div className="wrap">

```

```

    <SliderComponent
      id="slider"
      min={min}
      max={max}
      value={value}
      step={step}
      tooltip={tooltip}
      ticks={ticks}
      showButtons={true}
      tooltipChange={tooltipChangeHandler.bind(this) as any}
      renderingTicks={renderingTicksHandler.bind(this) as any}
    />
  </div>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Numeric range slider in React Range slider component

The numeric values can be formatted into different decimal digits or fixed number of whole numbers or to represent the units. The Numeric processing is demonstrated below.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  // Slider ticks customization
  let ticks01 = {
    placement: "After",
    format: "##.## Km",
    largeStep: 20,
    smallStep: 10,
    showSmallTicks: true
  };
  let tooltip01 = { isVisible: true, format: "##.## Km" };
  let ticks02 = {
    placement: "After",
    format: "##.##00",
    largeStep: 0.02,
    smallStep: 0.01,
    showSmallTicks: true
  };
  let tooltip02 = { isVisible: true, format: "##.##00" };
  let tooltip03 = { isVisible: true, format: "00##" };
  let ticks03 = {
    placement: "After",
    format: "00##",
    largeStep: 20,
    smallStep: 10,
    showSmallTicks: true
  };
  return (
    <div id="container">
      <div className="wrap">

```

```

    <div className="label">Slider formatted with unit
representation</div>
    <SliderComponent id="slider" min={0} max={100} value={30} step={1}
tooltip={tooltip01} ticks={ticks01}/>
  </div>
  <div className="wrap">
    <div className="label">Slider formatted with three decimal
specifiers</div>
    <SliderComponent id="slider1" min={0.1} max={0.2} value={0.13}
step={0.01} tooltip={tooltip02} ticks={ticks02}/>
  </div>
  <div className="wrap">
    <div className="label">Slider formatted with two leading zeros</div>
    <SliderComponent id="slider2" min={0} max={100} value={30} step={1}
tooltip={tooltip03} ticks={ticks03}/>
  </div>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  // Slider ticks customization
  let ticks01: TicksDataModel = {
    placement: "After",
    format: "##.## Km",
    largeStep: 20,
    smallStep: 10,
    showSmallTicks: true
  };
  let tooltip01: TooltipDataModel = { isVisible: true, format: "##.## Km" };
  let ticks02: TicksDataModel = {
    placement: "After",
    format: "##.##00",
    largeStep: 0.02,
    smallStep: 0.01,
    showSmallTicks: true
  };
  let tooltip02: TooltipDataModel = { isVisible: true, format: "##.##00" };
  let tooltip03: TooltipDataModel = { isVisible: true, format: "00##" };
  let ticks03: TicksDataModel = {
    placement: "After",
    format: "00##",
    largeStep: 20,
    smallStep: 10,
    showSmallTicks: true
  };
  return (
    <div id="container">
      <div className="wrap">

```

```

    <div className="label">Slider formatted with unit
representation</div>
    <SliderComponent
      id="slider"
      min={0}
      max={100}
      value={30}
      step={1}
      tooltip={tooltip01}
      ticks={ticks01}
    />
  </div>
  <div className="wrap">
    <div className="label">Slider formatted with three decimal
specifiers</div>
    <SliderComponent
      id="slider1"
      min={0.1}
      max={0.2}
      value={0.13}
      step={0.01}
      tooltip={tooltip02}
      ticks={ticks02}
    />
  </div>
  <div className="wrap">
    <div className="label">Slider formatted with two leading zeros</div>
    <SliderComponent
      id="slider2"
      min={0}
      max={100}
      value={30}
      step={1}
      tooltip={tooltip03}
      ticks={ticks03}
    />
  </div>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Customize slider bar in React Range slider component

Slider appearance can be customized through CSS. By overriding the slider CSS classes, you can customize the slider bar.

The slider bar can be customized with different themes. By default, slider have class name e-slider-track for bar. The

class can be overridden with our own color values like the following code snippet.

```

`css
.e-control.e-slider .e-slider-track .e-range {

```

```

background: linear-gradient(left, #e1451d 0, #fdff47 17%, #86f9fe 50%, #2900f8 65%, #6e00f8 74%,
#e33df9 83%, #e14423 100%);
}
,

`ts
function change(args: SliderChangeEventArgs) {
if (args.value > 0 && args.value <= 25) {
// Change handle and range bar color to green when
(sliderHandle as HTMLElement).style.backgroundColor = 'green';
(sliderTrack as HTMLElement).style.backgroundColor = 'green';
} else if (args.value > 25 && args.value <= 50) {
// Change handle and range bar color to royal blue
(sliderHandle as HTMLElement).style.backgroundColor = 'royalblue';
(sliderTrack as HTMLElement).style.backgroundColor = 'royalblue';
} else if (args.value > 50 && args.value <= 75) {
// Change handle and range bar color to dark orange
(sliderHandle as HTMLElement).style.backgroundColor = 'darkorange';
(sliderTrack as HTMLElement).style.backgroundColor = 'darkorange';
} else if (args.value > 75 && args.value <= 100) {
// Change handle and range bar color to red
(sliderHandle as HTMLElement).style.backgroundColor = 'red';
(sliderTrack as HTMLElement).style.backgroundColor = 'red';
}
}
,

```

You can also apply background color for a certain range depending upon slider values, using change event.

INDEX.JSX

```

{% raw %}
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let defaultObj = null;
  let sliderTrack = null;
  let sliderHandle = null;
  function changeEvent(args) {
    if (args.value > 0 && args.value <= 25) {

```

```

        // Change handle and range bar color to green when
        sliderHandle.style.backgroundColor = "green";
        sliderTrack.style.backgroundColor = "green";
    }
    else if (args.value > 25 && args.value <= 50) {
        // Change handle and range bar color to royal blue
        sliderHandle.style.backgroundColor = "royalblue";
        sliderTrack.style.backgroundColor = "royalblue";
    }
    else if (args.value > 50 && args.value <= 75) {
        // Change handle and range bar color to dark orange
        sliderHandle.style.backgroundColor = "darkorange";
        sliderTrack.style.backgroundColor = "darkorange";
    }
    else if (args.value > 75 && args.value <= 100) {
        // Change handle and range bar color to red
        sliderHandle.style.backgroundColor = "red";
        sliderTrack.style.backgroundColor = "red";
    }
}
function created() {
    sliderTrack = defaultObj.element.querySelector(".e-range");
    sliderHandle = defaultObj.element.querySelector(".e-handle");
    sliderHandle.style.backgroundColor = "green";
    sliderTrack.style.backgroundColor = "green";
}
return (<div id="container">
    <div className="col-lg-12 control-section">
        <div className="slider-content-wrapper">
            <div className="slider_container">
                <div className="slider-labeltext slider_userselect">Height</div>
                <SliderComponent id="height_slider" min={0} max={100}
value={30}/>
            </div>
            <div className="slider_container">
                <div className="slider-labeltext slider_userselect">Gradient
color</div>
                <SliderComponent id="gradient_slider" type="MinRange" min={0}
max={100} value={30}/>
            </div>
            <div className="slider_container">
                <div className="slider-labeltext slider_userselect">
                    Dynamic thumb and selection bar color
                </div>
                <SliderComponent id="dynamic_color_slider" ref={slider => {
                    defaultObj = slider;
                }} type="MinRange" min={0} max={100} value={30}
change={changeEvent.bind(this)} created={created.bind(this)}/>
            </div>
        </div>
    </div>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
{% endraw %}

```

INDEX.TSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent, SliderChangeEventArgs } from '@syncfusion/ej2-react-inputs';
function App() {
  let defaultObj: SliderComponent = null as any;
  let sliderTrack: HTMLElement = null as any;
  let sliderHandle: HTMLElement = null as any;
  function changeEvent(args: SliderChangeEventArgs): void {
    if (args.value > 0 && args.value <= 25) {
      // Change handle and range bar color to green when
      sliderHandle.style.backgroundColor = "green";
      sliderTrack.style.backgroundColor = "green";
    } else if (args.value > 25 && args.value <= 50) {
      // Change handle and range bar color to royal blue
      sliderHandle.style.backgroundColor = "royalblue";
      sliderTrack.style.backgroundColor = "royalblue";
    } else if (args.value > 50 && args.value <= 75) {
      // Change handle and range bar color to dark orange
      sliderHandle.style.backgroundColor = "darkorange";
      sliderTrack.style.backgroundColor = "darkorange";
    } else if (args.value > 75 && args.value <= 100) {
      // Change handle and range bar color to red
      sliderHandle.style.backgroundColor = "red";
      sliderTrack.style.backgroundColor = "red";
    }
  }
  function created(): void {
    sliderTrack = defaultObj.element.querySelector(".e-range") as any;
    sliderHandle = defaultObj.element.querySelector(".e-handle") as any;
    sliderHandle.style.backgroundColor = "green";
    sliderTrack.style.backgroundColor = "green";
  }
  return (
    <div id="container">
      <div className="col-lg-12 control-section">
        <div className="slider-content-wrapper">
          <div className="slider_container">
            <div className="slider-labeltext slider_userselect">Height</div>
            <SliderComponent id="height_slider" min={0} max={100} value={30} />
          </div>
          <div className="slider_container">
            <div className="slider-labeltext slider_userselect">Gradient
color</div>
            <SliderComponent id="gradient_slider" type="MinRange" min={0}
max={100} value={30} />
          </div>
          <div className="slider_container">
            <div className="slider-labeltext slider_userselect">
Dynamic thumb and selection bar color
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}
```

```

        <SliderComponent
          id="dynamic_color_slider"
          ref={slider => {
            defaultObj = slider as any;
          }}
          type="MinRange"
          min={0}
          max={100}
          value={30}
          change={changeEvent.bind(this) as any}
          created={created.bind(this)}
        />
      </div>
    </div>
  </div>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
[% endraw %]

```

Customize slider limits in React Range slider component

Slider appearance can be customized via CSS. By overriding the slider CSS classes, the slider limit bar can be customized.

```

`css
.e-slider-container.e-horizontal .e-limits {
background-color: rgba(69, 100, 233, 0.46);
}
`

```

Here, the limit bar is customized with different background color. By default, the slider has class `e-limits` for limits bar.

You can override the class with our own color values as given in the following code snippet.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let defaultObj;
  let value = [30, 70];
  let ticks = { placement: 'Before', largeStep: 20, smallStep: 5,
showSmallTicks: true };
  // Initialize tooltip with placement and showOn
  let tooltip = { isVisible: true, placement: 'Before', showOn: 'Focus' };
  // Set the limit values for the minrange slider
  let minLimits = { enabled: true, minStart: 10, minEnd: 40 };
  // Set the limit values for the range slider
  let rangeLimits = { enabled: true, minStart: 10, minEnd: 40, maxStart:
60, maxEnd: 90 };

```



```

    return (<div id='container'>
      <div className="content-wrapper">
        <div className='sliderwrap'>
          <label className="userselect">MinRange Slider With Limits</label>
          <SliderComponent id='minrange' type='MinRange' min={0} max={100}
value={30} ticks={ticks} tooltip={tooltip} limits={minLimits}/>
        </div>
        <div className='sliderwrap'>
          <label className="userselect">Range Slider With Limits</label>
          <SliderComponent id='range' type='Range' min={0} max={100}
value={value} ticks={ticks} tooltip={tooltip} limits={rangeLimits}/>
        </div>
      </div>
    </div>);
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent, SliderChangeEventArgs } from '@syncfusion/ej2-react-inputs';
function App() {
  let defaultObj: SliderComponent;
  let value: number[] = [30, 70];
  let ticks: object = { placement: 'Before', largeStep: 20, smallStep: 5,
showSmallTicks: true };
  // Initialize tooltip with placement and showOn
  let tooltip: object = { isVisible: true, placement: 'Before', showOn:
'Focus' };
  // Set the limit values for the minrange slider
  let minLimits: object = { enabled: true, minStart: 10, minEnd: 40 };
  // Set the limit values for the range slider
  let rangeLimits: object = { enabled: true, minStart: 10, minEnd: 40,
maxStart: 60, maxEnd: 90 };
  return (
    <div id='container'>
      <div className="content-wrapper">
        <div className='sliderwrap'>
          <label className="userselect">MinRange Slider With Limits</label>
          <SliderComponent id='minrange'
type='MinRange'
min={0} max={100} value={30} ticks={ticks} tooltip={tooltip}
limits={minLimits} />
        </div>
        <div className='sliderwrap'>
          <label className="userselect">Range Slider With Limits</label>
          <SliderComponent id='range' type='Range'
min={0} max={100} value={value} ticks={ticks} tooltip={tooltip}
limits={rangeLimits} />
        </div>
      </div>
    </div>
  );
}

```

```

}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Customize slider ticks label in React Range slider component

Slider view can be customized via CSS. By overriding the slider CSS classes, you can customize the ticks. The ticks in slider allows you to easily identify the current value/values of the slider. It contains [smallStep](#) and [largeStep](#). By default, slider has class `e-tick` for slider ticks. You can override the class as per your requirement. Refer to the following code snippet to render ticks.

```

`css

.e-scale .e-tick.e-custom::before {
content: '\e967';
position: absolute;
}
`

`css
ticks_slider .e-scale :nth-child(1)::before {
color: red;
}
`

```

Here, the color for rendered ticks has been applied through `nth-child(childnumber)`. *The color is applied to the value of the childnumber* in the slider.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  let value = [30, 70];
  let icon = { placement: "Before", largeStep: 20 };
  let custom = { placement: "Both", largeStep: 20, smallStep: 5 };
  function iconTicks(args) {
    if (args.tickElement.classList.contains("e-large")) {
      args.tickElement.classList.add("e-custom");
    }
  }
  function customTicks(args) {
    let li = args.ticksWrapper.getElementsByClassName("e-large");
    let remarks = ["Very Poor", "Poor", "Average", "Good", "Very Good", "Excellent"];
    for (let i = 0; i < li.length; ++i) {
      li[i].querySelectorAll(".e-tick-both")[1].innerText = remarks[i];
    }
  }
  return (<div id="container">
    <div className="col-lg-12 control-section">

```

```

        <div className="slider-content-wrapper">
            <div className="slider_container" id="slider_wrapper">
                <div className="slider_labelText userselect">Dynamic ticks
color</div>
                <SliderComponent id="ticks_slider" type="MinRange" min={0}
max={100} step={5} value={30} ticks={icon}
renderingTicks={iconTicks.bind(this)}>/>
            </div>
            <div className="slider_container">
                <div className="slider_labelText userselect">Ticks with
legends</div>
                <SliderComponent id="slider" type="MinRange" min={0} max={100}
value={value} ticks={custom} renderedTicks={customTicks.bind(this)}>/>
            </div>
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent, SliderTickRenderedEventArgs, SliderTickEventArgs }
from '@syncfusion/ej2-react-inputs';
function App() {
    let value: number[] = [30, 70];
    let icon: TicksDataModel = { placement: "Before", largeStep: 20 };
    let custom: TicksDataModel = { placement: "Both", largeStep: 20, smallStep:
5 };
    function iconTicks(args: SliderTickEventArgs): void {
        if (args.tickElement.classList.contains("e-large")) {
            args.tickElement.classList.add("e-custom");
        }
    }
    function customTicks(args: SliderTickRenderedEventArgs): void {
        let li: any = args.ticksWrapper.getElementsByClassName("e-large");
        let remarks: any = ["Very Poor", "Poor", "Average", "Good", "Very Good",
"Excellent"];
        for (let i: number = 0; i < li.length; ++i) {
            (li[i].querySelectorAll(".e-tick-both")[1] as HTMLElement).innerText =
remarks[i];
        }
    }
    return (
        <div id="container">
            <div className="col-lg-12 control-section">
                <div className="slider-content-wrapper">
                    <div className="slider_container" id="slider_wrapper">
                        <div className="slider_labelText userselect">Dynamic ticks
color</div>
                        <SliderComponent
                            id="ticks_slider"
                            type="MinRange"

```

```

        min={0}
        max={100}
        step={5}
        value={30}
        ticks={icon}
        renderingTicks={iconTicks.bind(this) as any}
      />
    </div>
    <div className="slider_container">
      <div className="slider_labelText userselect">Ticks with
legends</div>
      <SliderComponent
        id="slider"
        type="MinRange"
        min={0}
        max={100}
        value={value}
        ticks={custom}
        renderedTicks={customTicks.bind(this) as any}
      />
    </div>
  </div>
</div>
</div>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Customize slider thumb in React Range slider component

Slider appearance can be customized through CSS. By overriding the slider CSS classes, you can customize the thumb. By default, slider has unique class `e-handle` for slider thumb. You can override the following class as per your requirement.

```

`css
.e-control.e-slider .e-handle {
background-image: url('https://ej2.syncfusion.com/demos/src/slider/images/thumb.png');
background-color: transparent;
height: 25px;
width: 25px;
}

square_slider.e-control.e-slider .e-handle {
border-radius: 0%;
background-color: #f9920b;
border: 0;
}

```

```

circle_slider.e-control.e-slider .e-handle {
border-radius: 50%;

background-color: #f9920b;

border: 0;
}

oval_slider.e-control.e-slider .e-handle {
height: 25px;

width: 8px;

top: 3px;

border-radius: 15px;

background-color: #f9920b;
}

```

Here, in the sample, the slider thumb has been customized to square, circle, oval shapes, and background image has also been customized.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    return (<div id='container'>
        <div className="col-lg-12 control-section">
            <div className="slider-content-wrapper">
                <div className="slider_container">
                    <div className="labelText slider-userselect">Square</div>
                    <SliderComponent id='square_slider' min={0} max={100}
value={30}/>
                </div>
                <div className="slider_container">
                    <div className="labelText slider-userselect">Circle</div>
                    <SliderComponent id='circle_slider' min={0} max={100}
value={30}/>
                </div>
                <div className="slider_container">
                    <div className="labelText slider-userselect">Oval</div>
                    <SliderComponent id='oval_slider' min={0} max={100}
value={30}/>
                </div>
                <div className="slider_container">
                    <div className="labelText slider-userselect">Custom image</div>
                    <SliderComponent id='image_slider' min={0} max={100}
value={30}/>
                </div>
            </div>
        </div>
    </div>);
}

```

```
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent, SliderTickRenderedEventArgs, SliderTickEventArgs }
from '@syncfusion/ej2-react-inputs';
function App() {
    return (
        <div id='container'>
            <div className="col-lg-12 control-section">
                <div className="slider-content-wrapper">
                    <div className="slider_container">
                        <div className="labelText slider-userselect">Square</div>
                        <SliderComponent id='square_slider' min={0} max={100}
value={30} />
                    </div>
                    <div className="slider_container">
                        <div className="labelText slider-userselect">Circle</div>
                        <SliderComponent id='circle_slider' min={0} max={100}
value={30} />
                    </div>
                    <div className="slider_container">
                        <div className="labelText slider-userselect">Oval</div>
                        <SliderComponent id='oval_slider' min={0} max={100} value={30}
/>>
                    </div>
                    <div className="slider_container">
                        <div className="labelText slider-userselect">Custom image</div>
                        <SliderComponent id='image_slider' min={0} max={100} value={30}
/>>
                    </div>
                </div>
            </div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Form slider with formvalidator in React Range slider component

The Slider component can be validated using our [FormValidator](#). The following steps walk-through slider validation.

- Render slider component inside a form.
- Bind [changed](#) event in the slider component to validate the slider value when the value changes.
- Initialize and render FormValidator for the form using form ID.

```
`ts
```

```
// Initialize Form validation
```

```
let formObj: FormValidator;
formObj = new FormValidator("#formId", options);
`ts
```

- Set the required property in the FormValidator [rules](#) collection. Here, the [min](#) property of slider that sets the minimum value in the slider component is set, and it has hidden input as enable `validateHidden` property is set to true.

```
// Slider element
<div id="default" name="slider"></div>
// sets required property in the FormValidator rules collection
let options: FormValidatorModel = {
rules: {
'default': {
validateHidden: true,
min: [6, "You must select value greater than 5"]
}
}
};
`ts
```

Form validation is done either by ID or name value of the slider component. Above ID of the slider is used to validate it.

Using slider name: Render slider with name attribute. In the following code snippet, name attribute value of slider is used for form validation.

```
`ts
// Slider element
<div id="default" name="slider"></div>
// sets required property in the FormValidator rules collection
let options: FormValidatorModel = {
rules: {
'slider': {
validateHidden: true,
min: [6, "You must select value greater than 5"]
}
}
};
`ts
```

```

}
};
`

```

- Validate the form using [validate](#) method, and it validates the slider value with the defined rules collection and returns the result. If user selects the value less than the minimum value, form will not submit.

```

`ts
formObj.validate();
`

```

- Slider validation can be done during value changes in slider. Refer to the following code snippet.

```

`ts
// change event handler for slider
function onChanged(args: any) {
formObj.validate();
}
`

```

The **FormValidator** has following default validation rules, which are used to validate the Slider component.

Rules	Description	Example
-----	-----	-----
max	Slider component must have value less than or equal to max number	if max: 3 , 3 is valid and 4 is invalid
min	Slider component must have value greater than or equal to min number	if min: 4 , 5 is valid and 2 is invalid
regex	Slider component must have valid value in regex format	if regex: '/4/' , 4 is valid and 1 is invalid
range	Slider component must have value between range number	if range: [4,5] , 4 is valid and 6 is invalid

INDEX.JSX

```

{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
import { FormValidator } from '@syncfusion/ej2-inputs';
function App() {
  React.useEffect(() => {

```



```
        componentDidMount();
    }, []);
    let ticks = { placement: "Before", largeStep: 20, smallStep: 5,
showSmallTicks: true };
    let value = [30, 70];
    // sets required property in the FormValidator rules collection
    let minOptions = {
        rules: {
            "min-slider": {
                validateHidden: true,
                min: [40, "You must select value greater than or equal to
40"]
            }
        }
    };
    // sets required property in the FormValidator rules collection
    let maxOptions = {
        rules: {
            "max-slider": {
                validateHidden: true,
                max: [40, "You must select value less than or equal to 40"]
            }
        }
    };
    // sets required property in the FormValidator rules collection
    let valOptions = {
        rules: {
            "val-slider": {
                validateHidden: true,
                regex: [/40/, "You must select value equal to 40"]
            }
        }
    };
    // sets required property in the FormValidator rules collection
    let rangeOptions = {
        rules: {
            "range-slider": {
                validateHidden: true,
                range: [40, 80, "You must select values between 40 and 80"]
            }
        }
    };
    // sets required property in the FormValidator rules collection
    let customOptions = {
        rules: {
            "custom-slider": {
                validateHidden: true,
                range: [validateRange.bind(this), "You must select values
between 40 and 80"]
            }
        }
    };
    // Initialize Form validation
    let formMinObj = null;
    let formMaxObj = null;
    let formValObj = null;
    let formRangeObj = null;
```

```

let formCustomObj = null;
function componentDidMount() {
  formMinObj = new FormValidator("#formMinId", minOptions);
  formMaxObj = new FormValidator("#formMaxId", maxOptions);
  formValObj = new FormValidator("#formValId", valOptions);
  formRangeObj = new FormValidator("#formRangeId", rangeOptions);
  formCustomObj = new FormValidator("#formCustomId", customOptions);
}
function onMinChanged(args) {
  // validate the slider value in the form
  formMinObj.validate();
}
function onMaxChanged(args) {
  // validate the slider value in the form
  formMaxObj.validate();
}
function onValChanged(args) {
  // validate the slider value in the form
  formValObj.validate();
}
function onRangeChanged(args) {
  // validate the slider value in the form
  formRangeObj.validate();
}
function onCustomChanged(args) {
  // validate the slider value in the form
  formCustomObj.validate();
}
let SliderCustomObj = null;
function validateRange(args) {
  return (SliderCustomObj.value[0] >= 40 &&
    SliderCustomObj.value[1] <= 80);
}
return (<div id="container">
  <div className="col-lg-12 control-section">
    <div className="content-wrapper">
      <div className="form-title">
        <span>Min</span>
      </div>
      <form id="formMinId" className="form-horizontal">
        <div className="form-group">
          <div className="e-float-input">
            <SliderComponent id="min-slider" name="min-slider"
type="MinRange" value={30} ticks={ticks} changed={onMinChanged.bind(this)} />
          </div>
        </div>
      </form>
      <div className="form-title">
        <span>Max</span>
      </div>
      <form id="formMaxId" className="form-horizontal">
        <div className="form-group">
          <div className="e-float-input">
            <SliderComponent id="max-slider" name="max-slider"
type="MinRange" value={30} ticks={ticks} changed={onMaxChanged.bind(this)} />
          </div>
        </div>
      </form>
    </div>
  </div>
</div>

```

```

    </form>
    <div className="form-title">
      <span>Value</span>
    </div>
    <form id="formValId" className="form-horizontal">
      <div className="form-group">
        <div className="e-float-input">
          <SliderComponent id="val-slider" name="val-slider"
type="MinRange" value={30} ticks={ticks} changed={onValChanged.bind(this)} />
        </div>
      </div>
    </form>
    <div className="form-title">
      <span>Range</span>
    </div>
    <form id="formRangeId" className="form-horizontal">
      <div className="form-group">
        <div className="e-float-input">
          <SliderComponent id="range-slider" name="range-slider"
type="MinRange" value={30} ticks={ticks}
changed={onRangeChanged.bind(this)} />
        </div>
      </div>
    </form>
    <div className="form-title">
      <span>Custom</span>
    </div>
    <form id="formCustomId" className="form-horizontal">
      <div className="form-group">
        <div className="e-float-input">
          <SliderComponent id="custom-slider" name="custom-slider"
type="Range" value={value} ticks={ticks} changed={onCustomChanged.bind(this)}
ref={slider => {
      SliderCustomObj = slider;
    }} />
        </div>
      </div>
    </form>
  </div>
</div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
function App() {
  React.useEffect(() => {

```

```

    componentDidMount();
  }, []);
  let ticks: object = { placement: "Before", largeStep: 20, smallStep: 5,
showSmallTicks: true };
  let value: number[] = [30, 70];
  // sets required property in the FormValidator rules collection
  let minOptions: FormValidatorModel = {
    rules: {
      "min-slider": {
        validateHidden: true,
        min: [40, "You must select value greater than or equal to 40"]
      }
    }
  };
  // sets required property in the FormValidator rules collection
  let maxOptions: FormValidatorModel = {
    rules: {
      "max-slider": {
        validateHidden: true,
        max: [40, "You must select value less than or equal to 40"]
      }
    }
  };
  // sets required property in the FormValidator rules collection
  let valOptions: FormValidatorModel = {
    rules: {
      "val-slider": {
        validateHidden: true,
        regex: [/40/, "You must select value equal to 40"]
      }
    }
  };
  // sets required property in the FormValidator rules collection
  let rangeOptions: FormValidatorModel = {
    rules: {
      "range-slider": {
        validateHidden: true,
        range: [40, 80, "You must select values between 40 and 80"]
      }
    }
  };
  // sets required property in the FormValidator rules collection
  let customOptions: FormValidatorModel = {
    rules: {
      "custom-slider": {
        validateHidden: true,
        range: [validateRange.bind(this), "You must select values between 40
and 80"]
      }
    }
  };
  // Initialize Form validation
  let formMinObj: FormValidator = null as any;
  let formMaxObj: FormValidator = null as any;
  let formValObj: FormValidator = null as any;
  let formRangeObj: FormValidator = null as any;
  let formCustomObj: FormValidator = null as any;

```

```

function componentDidMount() {
  formMinObj = new FormValidator("#formMinId", minOptions);
  formMaxObj = new FormValidator("#formMaxId", maxOptions);
  formValObj = new FormValidator("#formValId", valOptions);
  formRangeObj = new FormValidator("#formRangeId", rangeOptions);
  formCustomObj = new FormValidator("#formCustomId", customOptions);
}
function onMinChanged(args: any): void {
  // validate the slider value in the form
  formMinObj.validate();
}
function onMaxChanged(args: any): void {
  // validate the slider value in the form
  formMaxObj.validate();
}
function onValChanged(args: any): void {
  // validate the slider value in the form
  formValObj.validate();
}
function onRangeChanged(args: any) {
  // validate the slider value in the form
  formRangeObj.validate();
}
function onCustomChanged(args: any) {
  // validate the slider value in the form
  formCustomObj.validate();
}
let SliderCustomObj: SliderComponent = null as any;
function validateRange(args: any) {
  return (
    (SliderCustomObj.value as number[])[0] >= 40 &&
    (SliderCustomObj.value as number[])[1] <= 80
  );
}
return (
  <div id="container">
    <div className="col-lg-12 control-section">
      <div className="content-wrapper">
        <div className="form-title">
          <span>Min</span>
        </div>
        <form id="formMinId" className="form-horizontal">
          <div className="form-group">
            <div className="e-float-input">
              <SliderComponent
                id="min-slider"
                name="min-slider"
                type="MinRange"
                value={30}
                ticks={ticks}
                changed={onMinChanged.bind(this) }
              />
            </div>
          </div>
        </form>
        <div className="form-title">
          <span>Max</span>

```

```

</div>
<form id="formMaxId" className="form-horizontal">
  <div className="form-group">
    <div className="e-float-input">
      <SliderComponent
        id="max-slider"
        name="max-slider"
        type="MinRange"
        value={30}
        ticks={ticks}
        changed={onMaxChanged.bind(this)}
      />
    </div>
  </div>
</form>
<div className="form-title">
  <span>Value</span>
</div>
<form id="formValId" className="form-horizontal">
  <div className="form-group">
    <div className="e-float-input">
      <SliderComponent
        id="val-slider"
        name="val-slider"
        type="MinRange"
        value={30}
        ticks={ticks}
        changed={onValChanged.bind(this)}
      />
    </div>
  </div>
</form>
<div className="form-title">
  <span>Range</span>
</div>
<form id="formRangeId" className="form-horizontal">
  <div className="form-group">
    <div className="e-float-input">
      <SliderComponent
        id="range-slider"
        name="range-slider"
        type="MinRange"
        value={30}
        ticks={ticks}
        changed={onRangeChanged.bind(this)}
      />
    </div>
  </div>
</form>
<div className="form-title">
  <span>Custom</span>
</div>
<form id="formCustomId" className="form-horizontal">
  <div className="form-group">
    <div className="e-float-input">
      <SliderComponent
        id="custom-slider"

```

```

        name="custom-slider"
        type="Range"
        value={value}
        ticks={ticks}
        changed={onCustomChanged.bind(this)}
        ref={slider => {
            SliderCustomObj = slider as any;
        }}
    />
</div>
</div>
</form>
</div>
</div>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
{% endraw %}

```

Show slider from hidden state in React Range slider component

This section demonstrates how-to render the Slider component in hidden state and make it visible in button click. We can initialize Slider in hidden state by setting the display as none.

In the sample, by clicking on the button, we can make the Slider visible from hidden state, and we must also call the [refresh](#) method of the Slider to render it properly based on its original dimensions.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    let sliderInstance;
    let min = new Date(2013, 6, 13, 11).getTime();
    let max = new Date(2013, 6, 13, 17).getTime();
    let value = new Date(2013, 6, 13, 13).getTime();
    let step = 3600000;
    let showButtons = true;
    // Slider ticks customization
    let ticks = { placement: "After", largeStep: 2 * 3600000 };
    let tooltip = { placement: "Before", isVisible: true };
    function renderingTicksHandler(args) {
        let totalMiliSeconds = Number(args.value);
        let custom = { hour: "2-digit", minute: "2-digit" };
        args.text = new Date(totalMiliSeconds).toLocaleTimeString("en-us",
custom);
    }
    function tooltipChangeHandler(args) {
        let totalMiliSeconds = Number(args.text);
        let custom = { hour: "2-digit", minute: "2-digit" };
        args.text = new Date(totalMiliSeconds).toLocaleTimeString("en-us",
custom);
    }
    function onClick() {

```

```

        const slider = document.getElementById("case");
        if (slider) {
            slider.style.display = "block";
            sliderInstance.refresh();
        }
    }
    return (<div id="container">
        <button onClick={onClick.bind(this)}>Button</button>
        <div id="case" className="wrap">
            <SliderComponent ref={t => (sliderInstance = t)} min={min} max={max}
value={value} step={step} tooltip={tooltip} ticks={ticks} showButtons={true}
tooltipChange={tooltipChangeHandler.bind(this)}
renderingTicks={renderingTicksHandler.bind(this)} />
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { SliderComponent, TicksDataModel, TooltipDataModel,
SliderTickEventArgs, SliderTooltipEventArgs } from '@syncfusion/ej2-react-
inputs';
function App() {
    let sliderInstance: SliderComponent;
    let min = new Date(2013, 6, 13, 11).getTime();
    let max = new Date(2013, 6, 13, 17).getTime();
    let value = new Date(2013, 6, 13, 13).getTime();
    let step = 3600000;
    let showButtons = true;
    // Slider ticks customization
    let ticks: TicksDataModel = { placement: "After", largeStep: 2 * 3600000 };
    let tooltip: TooltipDataModel = { placement: "Before", isVisible: true };
    function renderingTicksHandler(args: SliderTickEventArgs) {
        let totalMiliSeconds = Number(args.value);
        let custom = { hour: "2-digit", minute: "2-digit" };
        args.text = new Date(totalMiliSeconds).toLocaleTimeString("en-us",
custom);
    }
    function tooltipChangeHandler(args: SliderTooltipEventArgs) {
        let totalMiliSeconds = Number(args.text);
        let custom = { hour: "2-digit", minute: "2-digit" };
        args.text = new Date(totalMiliSeconds).toLocaleTimeString("en-us",
custom);
    }

    function onClick(){
        const slider = document.getElementById("case");
        if(slider){
            slider.style.display = "block";
            sliderInstance.refresh();
        }
    }
}

```



```

return (
  <div id="container">
    <button onClick={onClick.bind(this)}>Button</button>
    <div id="case" className="wrap">
      <SliderComponent
        ref={t=>(sliderInstance = t as SliderComponent)}
        min={min}
        max={max}
        value={value}
        step={step}
        tooltip={tooltip}
        ticks={ticks}
        showButtons={true}
        tooltipChange={tooltipChangeHandler.bind(this) as any}
        renderingTicks={renderingTicksHandler.bind(this) as any}
      />
    </div>
  </div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Reversible Range Slider in React

You can create a Range Slider rendered with values in reverse order by setting the [min](#) property to the maximum value and the [max](#) property to the minimum value. An example of how to achieve a reversible Range Slider is shown below

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
  // Slider ticks customization
  let ticks01 = {
    placement: "After",
    format: "##.## Km",
    largeStep: 20,
    smallStep: 10,
    showSmallTicks: true
  };
  let tooltip01 = { isVisible: true, format: "##.## Km" };
  let ticks02 = {
    placement: "After",
    format: "##.##00",
    largeStep: 0.02,
    smallStep: 0.01,
    showSmallTicks: true
  };
  let tooltip02 = { isVisible: true, format: "##.##00" };
  let tooltip03 = { isVisible: true, format: "00##" };
  let ticks03 = {
    placement: "After",
    format: "00##",
    largeStep: 20,

```

```

        smallStep: 10,
        showSmallTicks: true
    };
    return (<div id="container">
        <div className="wrap">
            <div className="label">Slider formatted with unit
representation</div>
            <SliderComponent id="slider" min={0} max={100} value={30} step={1}
tooltip={tooltip01} ticks={ticks01}/>
        </div>
        <div className="wrap">
            <div className="label">Slider formatted with three decimal
specifiers</div>
            <SliderComponent id="slider1" min={0.1} max={0.2} value={0.13}
step={0.01} tooltip={tooltip02} ticks={ticks02}/>
        </div>
        <div className="wrap">
            <div className="label">Slider formatted with two leading zeros</div>
            <SliderComponent id="slider2" min={0} max={100} value={30} step={1}
tooltip={tooltip03} ticks={ticks03}/>
        </div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { SliderComponent } from '@syncfusion/ej2-react-inputs';
function App() {
    // Slider ticks customization
    let ticks01: TicksDataModel = {
        placement: "After",
        format: "##.## Km",
        largeStep: 20,
        smallStep: 10,
        showSmallTicks: true
    };
    let tooltip01: TooltipDataModel = { isVisible: true, format: "##.## Km" };
    let ticks02: TicksDataModel = {
        placement: "After",
        format: "##.##00",
        largeStep: 0.02,
        smallStep: 0.01,
        showSmallTicks: true
    };
    let tooltip02: TooltipDataModel = { isVisible: true, format: "##.##00" };
    let tooltip03: TooltipDataModel = { isVisible: true, format: "00##" };
    let ticks03: TicksDataModel = {
        placement: "After",
        format: "00##",
        largeStep: 20,
        smallStep: 10,
        showSmallTicks: true
    };
}

```

```

    };
    return (
      <div id="container">
        <div className="wrap">
          <div className="label">Slider formatted with unit
representation</div>
          <SliderComponent
            id="slider"
            min={0}
            max={100}
            value={30}
            step={1}
            tooltip={tooltip01}
            ticks={ticks01}
          />
        </div>
        <div className="wrap">
          <div className="label">Slider formatted with three decimal
specifiers</div>
          <SliderComponent
            id="slider1"
            min={0.1}
            max={0.2}
            value={0.13}
            step={0.01}
            tooltip={tooltip02}
            ticks={ticks02}
          />
        </div>
        <div className="wrap">
          <div className="label">Slider formatted with two leading zeros</div>
          <SliderComponent
            id="slider2"
            min={0}
            max={100}
            value={30}
            step={1}
            tooltip={tooltip03}
            ticks={ticks03}
          />
        </div>
      </div>
    );
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById('element'));

```

Reversible order can be achieved with [Horizontal](#) orientation Slider by setting [enableRtl](#) as true.

Rating

Getting Started with React Rating Control

This section explains how to create a simple Rating and demonstrate the basic usage of the Rating component in a React environment.

Dependencies

The list of dependencies required to use the Rating component in your application is given below:

```
`js
|-- @syncfusion/ej2-react-inputs
|-- @syncfusion/ej2-react-base
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-popups
`,`
```

Installation and Configuration

You can use [Create-react-app](#) to setup the applications. To install `create-react-app` run the following command.

```
`bash
npm install -g create-react-app
`,`
```

Start a new project using create-react-app command as follows

```
`
create-react-app quickstart --scripts-version=react-scripts-ts
cd quickstart
`,`
```

```
create-react-app quickstart
cd quickstart
`,`
```

'react-scripts-ts' is used for creating React app with typescript.

Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) public registry.

To install Rating component, use the following command

```
`bash
npm install @syncfusion/ej2-react-inputs --save
`,`
```

Adding Rating component to the Application

To include the Rating component in your application import the `RatingComponent` from `ej2-react-inputs` package in `App.tsx`.

Add the Rating component in application as shown in below code example.

```

`ts
{ / Import the Rating./ }
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import './App.css';
{ / To render Rating. / }
function App() {
  return (
    <RatingComponent id='rating'></RatingComponent>
  );
}
export default App;
`

```

Adding CSS Reference

Import the Rating component's required CSS references as follows in `src/App.css`.

```

`css
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
`

```

Running the application

Run the application in the browser using the following command:

```

npm start
`

```

The following example shows a basic Rating component.

INDEX.JSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
  return (
    <div className='wrap'>
      <RatingComponent id='rating'></RatingComponent>
    </div>);
}
export default App;

```

```
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating'></RatingComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Value

You can set the rating value by using the [value](#) property.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <RatingComponent id='rating' value={3.0}></RatingComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3.0}></RatingComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Precision Modes in React Rating Component

You can use the [precision](#) property of the rating component to provide ratings with varying levels of precision.

The precision types of Rating are as follows:

- Full: The rating is increased in whole number increments. For example, if the current rating is 2, the next possible ratings are 3, 4, and so on.
- Half: The rating is increased in increments of 0.5 (half). For example, if the current rating is 2.5, the next possible ratings are 3, 3.5, 4, and so on.
- Quarter: The rating is increased in increments of 0.25 (quarter). For example, if the current rating is 3.75, the next possible ratings are 4, 4.25, 4.5, and so on.
- Exact: The rating is increased in increments of 0.1. For example, if the current rating is 3.9, the next possible ratings are 4, 4.1, 4.2, and so on.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent, PrecisionType } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <label>Full Precision</label><br />
        <RatingComponent id='rating1' value={3}
precision={PrecisionType.Full}></RatingComponent><br />
        <label>Half Precision</label><br />
        <RatingComponent id='rating2' value={2.5}
precision={PrecisionType.Half}></RatingComponent><br />
        <label>Quarter Precision</label><br />
        <RatingComponent id='rating3' value={3.75}
precision={PrecisionType.Quarter}></RatingComponent><br />
        <label>Exact Precision</label><br />
        <RatingComponent id='rating4' value={2.3}
precision={PrecisionType.Exact}></RatingComponent><br />
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent, PrecisionType } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (
```

```

        <div className='wrap'>
            <label>Full Precision</label><br/>
            <RatingComponent id='rating1' value={3} precision=
{PrecisionType.Full} ></RatingComponent><br/>
            <label>Half Precision</label><br/>
            <RatingComponent id='rating2' value={2.5} precision=
{PrecisionType.Half} ></RatingComponent><br/>
            <label>Quarter Precision</label><br/>
            <RatingComponent id='rating3' value={3.75} precision=
{PrecisionType.Quarter} ></RatingComponent><br/>
            <label>Exact Precision</label><br/>
            <RatingComponent id='rating4' value={2.3} precision=
{PrecisionType.Exact} ></RatingComponent><br/>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Labels in React Rating Component

You can use the [showLabel](#) property to display a label that shows the current value of the rating. When the `showLabel` property is set to `true`, a label will be displayed.

INDEX.JSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3}
showLabel={true}></RatingComponent>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3} showLabel= {true}
></RatingComponent>
        </div>
    );
}

```



```
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Label position

The rating component allows you to place the label on the top, bottom, left, or right side of the rating using the [labelPosition](#) property.

The following label positions are supported:

- Top: The label is placed on the top of the rating.
- Bottom: The label is placed on the bottom of the rating.
- Left: The label is placed on the left side of the rating.
- Right: The label is placed on the right side of the rating.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent, LabelPosition } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (
        <div className='wrap'>
            <label>Left Label Position</label><br />
            <RatingComponent id='rating1' value={3} showLabel={true}
labelPosition={LabelPosition.Left}></RatingComponent><br />
            <label>Right Label Position</label><br />
            <RatingComponent id='rating2' value={3} showLabel={true}
labelPosition={LabelPosition.Right}></RatingComponent><br />
            <label>Top Label Position</label><br />
            <RatingComponent id='rating3' value={3} showLabel={true}
labelPosition={LabelPosition.Top}></RatingComponent><br />
            <label>Bottom Label Position</label><br />
            <RatingComponent id='rating4' value={3} showLabel={true}
labelPosition={LabelPosition.Bottom}></RatingComponent><br />
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent, LabelPosition } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (
        <div className='wrap'>
            <label>Left Label Position</label><br/>
```

```

        <RatingComponent id='rating1' value={3} showLabel= {true}
labelPosition= {LabelPosition.Left} ></RatingComponent><br/>
        <label>Right Label Position</label><br/>
        <RatingComponent id='rating2' value={3} showLabel= {true}
labelPosition= {LabelPosition.Right}></RatingComponent><br/>
        <label>Top Label Position</label><br/>
        <RatingComponent id='rating3' value={3} showLabel= {true}
labelPosition= {LabelPosition.Top} ></RatingComponent><br/>
        <label>Bottom Label Position</label><br/>
        <RatingComponent id='rating4' value={3} showLabel= {true}
labelPosition= {LabelPosition.Bottom} ></RatingComponent><br/>
    </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Label template

You can use the [labelTemplate](#) tag directive to specify a custom template for the **Label** of the rating. The current value of the rating will be passed as the **value** property in the template context when building the content of the label. This allows you to include dynamic information about the rating in the template.

INDEX.JSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <RatingComponent id='rating' showLabel={true} value={3}
labelTemplate='<span>${value} out of 5</span>'></RatingComponent>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' showLabel={true} value={3}
labelTemplate='<span>${value} out of 5</span>' ></RatingComponent>
        </div>
    );
}

```

```
export default App;  
ReactDOM.render(<App />, document.getElementById('element'));
```

Tooltip in React Rating Component

The rating component supports tooltip to show additional information in rating items by setting the [showTooltip](#) property. If enabled, the tooltip appears when the user hovers over a rating item.

INDEX.JSX

```
// Import the Rating.  
import { RatingComponent } from '@syncfusion/ej2-react-inputs';  
import * as React from 'react';  
import * as ReactDOM from 'react-dom';  
// To render Rating.  
function App() {  
    return (<div className='wrap'>  
        <RatingComponent id='rating' showTooltip={true}  
value={3}></RatingComponent>  
        </div>);  
}  
export default App;  
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.  
import { RatingComponent } from '@syncfusion/ej2-react-inputs';  
import * as React from 'react';  
import * as ReactDOM from 'react-dom';  
// To render Rating.  
function App() {  
    return (  
        <div className='wrap'>  
            <RatingComponent id='rating' showTooltip={true} value={3}  
></RatingComponent>  
        </div>  
    );  
}  
export default App;  
ReactDOM.render(<App />, document.getElementById('element'));
```

Tooltip template

You can use the [tooltipTemplate](#) tag directive to specify a custom template for the **tooltip** of the rating. The current value of the rating will be passed as the **value** property in the template context when building the content of the tooltip. This allows you to include dynamic information about the rating in the template.

INDEX.JSX

```
// Import the Rating.  
import { RatingComponent } from '@syncfusion/ej2-react-inputs';  
import * as React from 'react';  
import * as ReactDOM from 'react-dom';
```

```
// To render Rating.
function App() {
  function tooltipTemplate(props) {
    if (props.value === 1) {
      return (<b>Angry</b>);
    }
    else if (props.value === 2) {
      return (<b>Sad</b>);
    }
    else if (props.value === 3) {
      return (<b>Neutral</b>);
    }
    else if (props.value === 4) {
      return (<b>Good</b>);
    }
    else {
      return (<b>Happy</b>);
    }
  }
  return (<div className='wrap'>
    <RatingComponent id='rating' showTooltip={true} value={3}
    tooltipTemplate={tooltipTemplate}></RatingComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
  function tooltipTemplate(props: any) {
    if(props.value === 1){
      return (<b>Angry</b>);}
    else if(props.value === 2){
      return (<b>Sad</b>);}
    else if(props.value === 3){
      return (<b>Neutral</b>);}
    else if(props.value === 4){
      return (<b>Good</b>);}
    else {
      return (<b>Happy</b>);}
  }

  return (
    <div className='wrap'>
      <RatingComponent id='rating' showTooltip={true} value={3}
      tooltipTemplate={tooltipTemplate} ></RatingComponent>
    </div>
  );
}
export default App;
```

```
ReactDOM.render(<App />, document.getElementById('element'));
```

Tooltip customization

You can customize the appearance of the tooltips using the `cssClass` property of the rating component and by defining the custom styles for tooltip elements like the below example.

You can find more information about customizing the appearance of the tooltip in the [Tooltip Customization](#) documentation.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <RatingComponent id='rating' showTooltip={true} value={3}
        cssClass='customtooltip'></RatingComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (
        <div className='wrap'>
            <RatingComponent id='rating' showTooltip={true} value={3}
            cssClass='customtooltip' ></RatingComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

STYLES.CSS

```
/* Represents the styles for loader */
#loader {
    color: #008cff;
    font-family: 'Helvetica Neue', 'calibiri';
    font-size: 14px;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
}
```

```

        width: 30%;
    }
    .wrap {
        margin: 50px auto;
        text-align: center;
    }
    /* To change the radius of the tooltip corners. */
    .customtooltip.e-tooltip-wrap {
        border-radius: 3px;
    }
    /* To change the size of the tooltip content. */
    .customtooltip.e-tooltip-wrap .e-tip-content {
        font-size: 14px;
    }
    /* To change the border color and width for tooltip. */
    .customtooltip.e-tooltip-wrap.e-popup {
        border: 2px solid #000000;
    }
    /* To change the color for arrow of the tooltip. */
    .customtooltip.e-tooltip-wrap .e-arrow-tip-inner.e-tip-bottom {
        border: 12px #9693
    }
    /* To change the top border color for arrow of the tooltip. */
    .customtooltip.e-tooltip-wrap .e-arrow-tip-outer.e-tip-bottom {
        border-top: 9.5px solid #000000;
    }
}

```

Selection in React Rating Component

The rating component allows users to rate something using a visual scale, and the selection state can be changed by the user clicking or tapping on the stars in the rating scale or through code. The rating component has a minimum value and a reset button, and provides customization options for the selected rating value and selection behavior.

INDEX.JSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3}></RatingComponent>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.

```

```
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3} ></RatingComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Min value

You can use the [min](#) property of the rating component to set the minimum possible rating value the user can select. If you set the `min` property to 2, then you will not be able to select a rating lower than 2.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <RatingComponent id='rating' min={2}></RatingComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' min={2} ></RatingComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Single selection

You can use the [enableSingleSelection](#) property of the rating component to select only one item at a time. When the `enableSingleSelection` property is set to `true`, only the selected item will be considered to be in the selected state, while all other items will be unselected.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <RatingComponent id='rating' value={3}
        enableSingleSelection={true}></RatingComponent>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3}
            enableSingleSelection={true} ></RatingComponent>
            </div>
        );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Show or hide reset button

You can reset the rating value to its default by using the [allowReset](#) property of the rating component. When the `allowReset` property is set to `true`, a reset button will be shown that allows the user to reset the rating value to its default.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <RatingComponent id='rating' value={3}
        allowReset={true}></RatingComponent>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```


INDEX.TSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3} allowReset={true}>
        </RatingComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Templates in React Rating Component

The rating component allows you to customize the appearance of the rating items using templates. You can use templates to specify a custom layout for the rating items, which can include any content you want. This allows you to create a more customized and interactive rating experience for the user.

The rating component supports below templates for item customization.

- [emptyTemplate](#)
- [fullTemplate](#)

Empty (unrated) symbol template

To customize the appearance of **unrated** items, you can use the **emptyTemplate** tag directive. It allows you to specify the desired custom content for the unrated items.

The **value** and **index** are available in the template context for accessing information about the un-rated item.

If the **fullTemplate** is not defined, the **emptyTemplate** will be used as the default for both rated and unrated items. You can apply custom styles to differentiate between the rated and unrated states of the items.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    function emptyTemplate() {
        return (<React.Fragment><span className='custom-font sf-rating-heart'></span></React.Fragment>);
    }
    return (<div className='wrap'>
```

```

        <RatingComponent id='rating' value={3}
emptyTemplate={emptyTemplate}></RatingComponent>
        </div>);
    }
    export default App;
    ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    function emptyTemplate() {
        return (
            <React.Fragment><span className='custom-font sf-rating-
heart'></span></React.Fragment>
        );
    }

    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3}
emptyTemplate={emptyTemplate} ></RatingComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

STYLES.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    font-family: 'Helvetica Neue', 'calibiri';
    font-size: 14px;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.wrap {
    margin: 50px auto;
    text-align: center;
}
.e-rating-container .custom-font {
    /* To add the background color for the font icon. */
    background: linear-gradient(to right, rgb(254,87,133,255) var(--rating-
value), transparent var(--rating-value));
    /* To clip the background to the icon (text) alone. */
    background-clip: text;
    -webkit-background-clip: text;
}

```

```

/* To make the background color visible instead of font color. */
-webkit-text-fill-color: transparent;
/* To provide a border for font icon. */
-webkit-text-stroke: 1px rgb(254, 87, 133, 255);
}
/* Represents the styles for icon */
@font-face {
  font-family: 'rating';
  src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMjluSfQAAAEoAAAAVmNtYXNlEudaAAABjAAAADhnbHlm4LiF
sgAAAcwAAAJsaGVhZCKCSVkaAADQAAAAANmhoZWEIUQQEAAAArAAAACRobXR4DAAAAAAAAAYAAAAAMb
G9jYQCMATYAAAEHAAAAACGlheHABDwCZAAABCAAAACBuYW1l75Kp8wAABDgAAAIzcG9zdDjyU90AAA
ZUAAAAANwABAAAAEAAAAAFwEAAAAAAD9AABAAAAAaAAAAAAAAAAAAAaBAAAAAQAA2T6Kh18PPPU
ACwQAAAAAAN+4AkEAAAAA37gCQQAAAAAD9APaAAAAACAACAAAAAAAAAAAAEAAAADAI0AAgAAAAAAAgAA
AAoACgAAAP8AAAAAAAAAAQAAZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAAAAAAAAA
AAAAAAAAAAAAAAAAAUGZFZABA5wHnAgQAAAAAXAQAAAAAABAAAAAABAAAAAQAAAAEAAAAAA
AAAgAAAAAUAAMAAQAAABQABAakAAAABAAEAAEAooC//8AAoCB//8AAAABAAQAAAACAAEAAAA
AAIwBNgABAAAAAAPzA9oAfAAAEw8WFR8PPw41Lx4jdDwvDw8GqAwMDAsKCgoJCAgIBwYGBQUEBAMC
AgEBAQECAwMEBQULFSMhOVJliOxTOSEdFg0IBQQDAwIBAQBAGIDBAQFBQYGBwgICakKCgoLDAwMD
AwMDQwMDQwZGBgYFxFVFBIRCAgGBwkLCwvNDg4PEBAQEREREhEODg4ODg4NA8IGBwcICakJCgoKCw
sMCwwNDA0MDQ0ODQ0ODQ0ODQ0NDRUimCtEX26P/V5FKycjFhQNDQ0ODQ0ODQ0NDgwNDQwNCwwMCwo
LCgoJCAkIBwcGBQUEAwMCAQEBCQYJCw4PERMKCGsMEQ8PDQ0LCwoICAYFBAMCAQEBAgIEBAUAAgAA
AAD9APFAAMajAAANzMRIwEPAXUXDwwRMzcfBDcXPwo9AS8FPwsvCDc1Pwg1LwU1Pw01LwkHJT8EN
S8LIw8BDK2tAfKCCgQBAQEBGCERERITIGkJKBAGIQc1Bx45k9sOBQgLDQsJBQMEAgIECQYCAQEBAw
4ECQgGBwMDAQEBQMDAwKCAQEDFgsFBAQDAwICAgQECgEBAQQKBwcGBQUEAwMBAQEBAUHCQUFBQY
R/q0PCQQDAgEBAwMKDBUDbwYMCw0HB1oBhwHeAQUDA3YfCgQsOh0bHBovCQgbDp6KAQEfAwEBAQIB
AQMGCGoMBggICAUICQgLBQQEBAUDBgMHCAGMCACIBwYGBgUFCQQCBgIEDakGBQYHCQkKCQgIBwsEA
gUDAgQEBAUFBgCHCACGBgYGCgkIBgICAQEBAUYxGRobDQ0MDQsiHjEEBAIEAQECAAEEgDeAAEAAA
AAAAAAAAQAAAAEAAAAAAAAEABgABAAEAAAAAAAAIABwAHAAEAAAAAAAAAMABgAOAAEAAAAAAAAQABgAUAAE
AAAAAAAAUACwAaAAEAAAAAAAAAYABgAlAAEAAAAAAAAoALAArAAEAAAAAAAAsAEgBXAAMAAQQJAAAAAgBp
AAAAAQJAAEADABrAAMAAQQJAAIADgB3AAMAAQQJAAMADACFAAMAAQQJAAQADACRAAMAAQQJAAUAF
gCdAAMAAQQJAAAYADACzAAMAAQQJAAoAWAC/AAMAAQQJAAAsAJAEXIHJhdGluZ1JlZ3VsYXJyYXRpbm
dyYXRpbmdWZXJzaW9uIDEuMHJhdGluZ0Zvb2Z2VzJhdGVkIHVzaW5nIFN5bmNmdXNpb24gTWV
0cm8gU3R1ZGlvd3d3LnN5bmNmdXNpb24uY29tACAACgBhAHQAaQBuAGcAUgBlAGcAdQBsaGEAcgBy
AGEAdABpAG4AZwByAGEAdABpAG4AZwBWBWAGUAcgBzAGkAbwBuACAAMQAuADAACgBhAHQAaQBuAGcAR
gBvAG4AdAAgAGcAZQBwAGUAcgBhAHQAQZQBkACAADQBzAGkAbgBnACAAUwB5AG4AYwBmAHUAcwBpAG
8AbgAgAE0AZQB0AHIAbwAgAFMADAB1AGQAaQBVaHcAdwB3AC4AcwB5AG4AYwBmAHUAcwBpAG8AbgA
uAGMABwBtAAAAAIAAAAAAAAAACgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAwECAQMBBAAFaGVhcnQF
dGhlbWIAAAA=) format('trueType');
  font-weight: normal;
  font-style: normal;
}
[class^="sf-rating-"], [class*=" sf-rating-"] {
  font-family: 'rating' !important;
  speak: none;
  font-style: normal;
  font-weight: normal;
  font-variant: normal;
  text-transform: none;
  line-height: 1;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
.sf-rating-heart:before {
  content: "\e702";
}

```

The current value of the rating item available in the template context as `value` and in the rating item element as CSS Variable (`--rating-value`) can be used to support precision in templates.

Full (rated) symbol template

To customize the appearance of **rated** items in the rating component, you can use the `fullTemplate` tag directive. This directive allows you to specify a custom layout for the rated items, which can include any content you desire.

The `value` and `index` are available in the template context for accessing information about the rated item.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
  function emptyTemplate() {
    return (<React.Fragment><span className='custom-font sf-icon-empty-star'></span></React.Fragment>);
  }
  function fullTemplate() {
    return (<React.Fragment><span className='custom-font sf-icon-fill-star'></span></React.Fragment>);
  }
  return (<div className='wrap'>
    <RatingComponent id='rating' value={3}
    emptyTemplate={emptyTemplate} fullTemplate={fullTemplate}></RatingComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
  function emptyTemplate() {
    return (
      <React.Fragment><span className='custom-font sf-icon-empty-star'></span></React.Fragment>
    );
  }
  function fullTemplate() {
    return (
      <React.Fragment><span className='custom-font sf-icon-fill-star'></span></React.Fragment>
    );
  }
}
```

```

    }

    return (
      <div className='wrap'>
        <RatingComponent id='rating' value={3}
          emptyTemplate={emptyTemplate} fullTemplate={fullTemplate} ></RatingComponent>
        </div>
      );
    }
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById('element'));

```

STYLES.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  font-family: 'Helvetica Neue', 'calibiri';
  font-size: 14px;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.wrap {
  margin: 50px auto;
  text-align: center;
}
.e-rating-container .custom-font {
  /* To change the icon font color. */
  color: rgb(255, 215, 0);
}
/* Represents the styles for icon */
@font-face {
  font-family: 'rating-template';
  src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKIAAAwAgTlMvMjltSfMAAAEoAAAAVmNtYXDNdEOdaAAABjAAAADhnbHlm+icD
jQAAAcwAAAE0aGVhZCK49ucAAADQAAAAANmhoZWEIUQQEAAAArAAAACRobXR4DAAAAAAAAAYAAAAAMb
G9jYQAcAJoaAAAEAAAAACG1heHABDwbKAAAABCAAAACBuYW1lmYExxgAAAwAAAAKFCg9zdCH169QAAA
WIAAAAAQAABAAAEAAAAAFwEAAAAAAD9AABAAAAAAAAAAAAAAAAAAAAAwABAAAAQAAGPX4jF8PPPU
ACwQAAAAAN/TWPsAAAAA39NY+wAAAAAD9AP0AAAAACAACAAAAAAAAAAAAEAAADAFgAAgAAAAAAgAA
AAoACgAAAP8AAAAAAAAAAQAAZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAAAAAAA
AAAAAAAAAAAAAAAAAUGZFZABA5wDnAQQAAAAAXAQAAAAAAAAABAAAAAAAAABAAAAQA AAAEAAAAAA
AAAgAAAAMAAAAUAMAAQAAABQABAakAAAABAAEAAEAAOCB//8AAOCa//8AAAABAAQAAAAABAAIAAAA
AABwAmgABAAAAAAP0A/QACQAAAQUTAYUFAXMlAwFn/qX6OwE1ATU7+v6lmQKsNf7//pasrAFqAQE1
AUgAAAIAAAAA/QD5AAdAFcAAAEfBAUPAXUTLwEjDwETNS8DJT8EJwMFDwQVHwIDBx8EMzclBRczP
wU1Az8CNS8DJQMvBisBDwUCYAICBgMHASCwBAMCGuoHCAjPggIDBLEBHgcGBgJiHXb+uQgHBgQBAG
TUHgECBAUHCAkIAQ4BDGcJBAQEbwQDHDMEAgMFBGf+tHYDAgMEBAQEBAQEBAQEAWICnwMDBgIDNBa
GBgYE/uCBAGKBAR0HBgYGsDQCBAYD3Fr+9jwDBACHBQgIB9T+twUIBwcEAWKVlQIBAGIFBwgJAUnU
BwgJCAcFBD0BCgQDBAICAgEBAGICBAMAAAAAEgDeAAEAAAAAAAAAAAAQAAAAEAAAAAAAAEADwBAAEAA
AAAAAIABwAQAAEAAAAAAAAAMADwAXAAEAAAAAAAAAQADwAmAAEAAAAAAAAAUACwA1AAEAAAAAAAAAYADwBAAA
EAAAAAAoALABPAAEAAAAAAAsAEgB7AAMAAQQJAAAAAgCNAAMAAQQJAAEAHGC PAAMAAQQJAAIADgC
tAAMAAQQJAAAMAHgC7AAMAAQQJAAQAHgDZAAMAAQQJAAUAFgD3AAMAAQQJAAAYAHgENAAMAAQQJAAoA
WAERAMAAQQJAAAsAJAGDIHJhdGluZy10ZW1wbGF0ZVZlZ3VsYXJyYXRpbmctdGVtcGxhdGVyYXRpb
mctdGVtcGxhdGVWZXJzaW9uIDEuMHJhdGluZy10ZW1wbGF0ZUZvbnQgZ2VuZXJhdGVkIHVzaW5nIF

```

```

N5bmNmdXNpb24gTWV0cm8gU3R1ZG1vd3d3LnN5bmNmdXNpb24uY29tACAacgBhAHQAaQBuAGcALQB
0AGUAbQBwAGwAYQB0AGUAUgBlAGcAdQBsAGEAcgByAGEAdABpAG4AZwAtAHQAZQBtAHAAbABhAHQA
ZQByAGEAdABpAG4AZwAtAHQAZQBtAHAAbABhAHQAZQBWAGUAcgBzAGkAbwBuACAAMQAuADAacgBhA
HQAAQBuAGcALQB0AGUAbQBwAGwAYQB0AGUARgBvAG4AdAAgAGcAZQBwAGUAcgBhAHQAZQBkACAAdQ
BzAGkAbgBnACAAUwB5AG4AYwBmAHUAcwBpAG8AbgAgAE0AZQB0AHIAbwAgAFMAdAB1AGQAaQBVAhc
AdwB3AC4AcwB5AG4AYwBmAHUAcwBpAG8AbgAuAGMAbwBtAAAAAIAAAAAAAACgAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAwECAQMBBAAJZmlsbC1zdGFyCmVtcHR5LXN0YXIAAA==)
format('true type');
font-weight: normal;
font-style: normal;
}
[class^="sf-icon-"],
[class*=" sf-icon-"] {
font-family: 'rating-template' !important;
speak: none;
font-style: normal;
font-weight: normal;
font-variant: normal;
text-transform: none;
line-height: 1;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
}
.sf-icon-fill-star:before {
content: "\e700";
}
.sf-icon-empty-star:before {
content: "\e701";
}

```

Using Emoji icon as rating symbol

You can use emojis of your choice as rating symbol by specifying them as template content within the `emptyTemplate` tag directive.

INDEX.JSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
function emptyTemplate(props) {
if (props.index === 0) {
return (<span className='angry emoji'>😡</span>);
}
else if (props.index === 1) {
return (<span className='disagree emoji'>😞</span>);
}
else if (props.index === 2) {
return (<span className='neutral emoji'>😐</span>);
}
else if (props.index === 3) {
return (<span className='agree emoji'>😄</span>);
}
}

```

```

        else {
            return (<span className='happy emoji'>😊</span>);
        }
    }
    return (<div className='wrap'>
        <RatingComponent id='rating' value={3}
        emptyTemplate={emptyTemplate} enableSingleSelection={true}
        enableAnimation={false}></RatingComponent>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    function emptyTemplate(props: any) {
        if(props.index===0) {
            return (<span className='angry emoji'>😡</span>); }
        else if(props.index===1) {
            return (<span className='disagree emoji'>😞</span>); }
        else if(props.index===2) {
            return (<span className='neutral emoji'>😐</span>); }
        else if(props.index===3) {
            return (<span className='agree emoji'>😊</span>); }
        else {
            return (<span className='happy emoji'>😊</span>); }
    }

    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3}
            emptyTemplate={emptyTemplate} enableSingleSelection={true}
            enableAnimation={false} ></RatingComponent>
            </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

STYLES.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    font-family: 'Helvetica Neue', 'calibiri';
    font-size: 14px;
    height: 40px;
    left: 45%;
}

```

```

    position: absolute;
    top: 45%;
    width: 30%;
  }
  .wrap {
    margin: 50px auto;
    text-align: center;
  }
  /* To change the color of an unselected rating item. */
  .e-rating-item-container:not(.e-rating-selected) .emoji {
    filter: grayscale(1);
  }
}

```

Using SVG icon as rating symbol

You can use SVG icons of your choice as rating symbol by specifying them as template content within the `emptyTemplate` and `fullTemplate` tag directive.

INDEX.JSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
  function emptyTemplate() {
    return (
      <svg width="35" height="25" className="e-rating-svg-icon">
        <rect width="35" height="25" fill="transparent" strokeWidth="2"
stroke="rgb(173,181,189)"/>
      </svg>);
  }
  function fullTemplate(props) {
    return (
      <svg width="35" height="25" className="e-rating-svg-icon">
        <defs>
          <linearGradient id={`grad${props.index}`} x1="0%" y1="0%"
x2="100%" y2="0%">
            <stop className="start" offset="0%"/>
            <stop className="end" offset="100%"/>
          </linearGradient>
        </defs>
        <rect width="35" height="25" fill={`url(#grad${props.index})`}
strokeWidth="2" stroke="rgb(173,181,189)"/>
      </svg>);
  }
  return (
    <div className="wrap">
      <RatingComponent id="rating" value={4}
emptyTemplate={emptyTemplate} fullTemplate={fullTemplate}
enableAnimation={false}></RatingComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX


```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
  function emptyTemplate() {
    return (
      <svg width="35" height="25" className="e-rating-svg-icon">
        <rect width="35" height="25" fill="transparent"
strokeWidth="2" stroke="rgb(173,181,189)"/>
      </svg>
    );
  }
  function fullTemplate(props: any) {
    return (
      <svg width="35" height="25" className="e-rating-svg-icon">
        <defs>
          <linearGradient id={`grad${props.index}`} x1="0%" y1="0%"
x2="100%" y2="0%">
            <stop className="start" offset="0%" />
            <stop className="end" offset="100%" />
          </linearGradient>
        </defs>
        <rect width="35" height="25" fill={`url(#grad${props.index})`}
strokeWidth="2" stroke="rgb(173,181,189)"/>
      </svg>
    );
  }
  return (
    <div className='wrap'>
      <RatingComponent id='rating' value={4}
emptyTemplate={emptyTemplate} fullTemplate={fullTemplate}
enableAnimation={false} ></RatingComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

STYLES.CSS

```
/* Represents the styles for loader */
#loader {
  color: #008cff;
  font-family: 'Helvetica Neue', 'calibiri';
  font-size: 14px;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.wrap {
  margin: 50px auto;
  text-align: center;
```

```

}
/* To change the size between items */
.e-rating-container .e-rating-item-container {
  padding: 0px;
}
/* To set the gradient color */
.e-rating-svg-icon #grad0 .start {
  stop-color: #FF0000;
}
.e-rating-svg-icon #grad0 .end,
.e-rating-svg-icon #grad1 .start {
  stop-color: #ff5101;
}
.e-rating-svg-icon #grad1 .end,
.e-rating-svg-icon #grad2 .start {
  stop-color: #ffc801;
}
.e-rating-svg-icon #grad2 .end,
.e-rating-svg-icon #grad3 .start {
  stop-color: #dbe300;
}
.e-rating-svg-icon #grad3 .end,
.e-rating-svg-icon #grad4 .start {
  stop-color: #8bc301;
}
.e-rating-svg-icon #grad4 .end {
  stop-color: #4eaa01;
}

```

Using PNG image as rating symbol

You can use PNG images of your choice as rating symbol by specifying them as template content within the `emptyTemplate` and `fullTemplate` tag directives.

INDEX.JSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
  function emptyTemplate() {
    return ();
  }
  function fullTemplate() {
    return ();
  }
  return (<div className='wrap'>
    <RatingComponent id='rating' value={4}
    emptyTemplate={emptyTemplate} fullTemplate={fullTemplate}>
    </RatingComponent>
    </div>);
}

```

```

}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
  function emptyTemplate() {
    return (
      
    );
  }
  function fullTemplate() {
    return (
      
    );
  }

  return (
    <div className='wrap'>
      <RatingComponent id='rating' value={4}
        emptyTemplate={emptyTemplate} fullTemplate={fullTemplate} >
      </RatingComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Events in React Rating Component

This section describes the rating events that will be triggered when appropriate actions are performed. The following events are available in the rating component.

beforeItemRender

The rating component triggers the [beforeItemRender](#) event before rendering each rating item. The [RatingItemEventArgs](#) passed as an event argument provides the details of the item to be rendered.

`ts

```
{/ Import the Rating. /}
```

```
import { RatingComponent, RatingItemEventArgs } from '@syncfusion/ej2-react-inputs';
```

```
import * as React from 'react';
```

```
import * as ReactDOM from 'react-dom';
```

```

{/ To render Rating./}
function App() {
  function beforeItemRender(args: RatingItemEventArgs){
    //Your required action here
  }
  return (
    <div className='wrap'>
      <RatingComponent id='rating' beforeItemRender={ beforeItemRender } ></RatingComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />,document.getElementById('element'));
`

```

created

The rating component triggers the [created](#) event when the rendering of the rating component is completed.

```

`ts
{/ Import the Rating. /}
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
{/ To render Rating./}
function App() {
  function created(){
    //Your required action here
  }
  return (
    <div className='wrap'>
      <RatingComponent id='rating' created={ created } ></RatingComponent>
    </div>
  );
}
export default App;

```

```
ReactDOM.render(<App />,document.getElementById('element'));  
,
```

onItemHover

The rating component triggers the [onItemHover](#) event when the rating item is hovered. The [RatingHoverEventArgs](#) passed as an event argument provides the details of the hovered item.

`ts

```
{/ Import the Rating. /}  
import { RatingComponent, RatingHoverEventArgs } from '@syncfusion/ej2-react-inputs';  
import * as React from 'react';  
import * as ReactDOM from 'react-dom';  
{/ To render Rating./}  
function App() {  
  function onItemHover(args: RatingHoverEventArgs){  
    //Your required action here  
  }  
  return (  
    <div className='wrap'>  
      <RatingComponent id='rating' onItemHover={ onItemHover } ></RatingComponent>  
    </div>  
  );  
}  
export default App;  
ReactDOM.render(<App />,document.getElementById('element'));  
,
```

valueChanged

The rating component triggers the [valueChanged](#) event when the value of the rating is changed. The [RatingChangedEventArgs](#) passed as an event argument provides the details when value is changed.

`ts

```
{/ Import the Rating. /}  
import { RatingComponent, RatingChangedEventArgs } from '@syncfusion/ej2-react-inputs';  
import * as React from 'react';  
import * as ReactDOM from 'react-dom';  
{/ To render Rating./}  
function App() {
```

```
function valueChanged(args: RatingChangedEventArgs){
//Your required action here
}
return (
<div className='wrap'>
<RatingComponent id='rating' valueChanged={ valueChanged } ></RatingComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />,document.getElementById('element'));
`
```

Below example demonstrates the valueChanged event of the Rating component.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    function valueChanged(args) {
        alert("Previous Value:" + args.previousValue + "\nValue:" +
args.value);
    }
    return (<div className='wrap'>
        <RatingComponent id='rating'
valueChanged={valueChanged}></RatingComponent>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent, RatingChangedEventArgs } from '@syncfusion/ej2-
react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    function valueChanged(args: RatingChangedEventArgs){
        alert("Previous Value:" +args.previousValue + "\nValue:" +args.value);
    }

    return (
```

```

        <div className='wrap'>
            <RatingComponent id='rating' valueChanged={ valueChanged }
        ></RatingComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Appearance in React Rating Component

You can also customize the appearance of rating component.

Items Count

You can specify the number of rating items using the [itemsCount](#) property.

INDEX.JSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <RatingComponent id='rating' itemsCount={8}
    value={3.0}></RatingComponent>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' itemsCount={8}
    value={3.0}></RatingComponent>
            </div>
        );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Disabled

You can disable the rating component by using the [disabled](#) property. When the `disabled` property is set to `true`, the rating component will be disabled and the user will not be able to interact with it and a disabled rating component may have a different visual appearance than an enabled one.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <RatingComponent id='rating' value={3}
disabled={true}></RatingComponent>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3} disabled={true}
></RatingComponent>
            </div>
        );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

Visible

You can use the [visible](#) property of the rating component to component the visibility of the component. When the [visible](#) property is set to `true`, the rating component will be visible on the page. When it is set to `false`, the component will be hidden.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    var rating;
    function visible() {
        rating.visible = !rating.visible;
    }
    return (<div className='wrap'>
        <button id="btn" onClick={visible}>Visible</button>
    );
}
```



```

    <RatingComponent id='rating' value={3} visible={true} ref={(scope)
=> { rating = scope; }}></RatingComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

// Import the Rating.
import { RatingComponent, Rating } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    var rating: Rating;
    function visible() {
        rating.visible = !rating.visible;
    }

    return (
        <div className='wrap'>
            <button id="btn" onClick={ visible }>Visible</button>
            <RatingComponent id='rating' value={3} visible={true}
ref={(scope) => { rating = scope; }} ></RatingComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

Read only

You can use the [readOnly](#) property of the rating component to make the component non-interactive and prevent the user from changing the rating value.

INDEX.JSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <RatingComponent id='rating' value={3}
readOnly={true}></RatingComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

// Import the Rating.

```

```
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3} readOnly={true}>
        </RatingComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

CssClass

You can customize the appearance of the rating component, such as by changing its colors, fonts, sizes, or other visual aspects by using the [cssClass](#) property.

Changing rating symbol border color

You can change the rating icon border color in rating component, you can use the `cssClass` property and set the `text-stroke` CSS property of `.e-rating-icon` to your desired border color.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <RatingComponent id='rating' value={3} cssClass='custom-
border'></RatingComponent>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3} cssClass='custom-border'
        ></RatingComponent>
        </div>
    );
}
```

```

}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

STYLES.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  font-family: 'Helvetica Neue', 'calibiri';
  font-size: 14px;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.wrap {
  margin: 50px auto;
  text-align: center;
}
.e-rating-container.custom-border .e-rating-item-list:hover .e-rating-item-
container .e-rating-icon,
.e-rating-container.custom-border .e-rating-item-container .e-rating-icon {
  /* To change rating symbol border color */
  -webkit-text-stroke: 2px #ae9e9d;
}

```

Changing rated/un-rated symbol fill color

You can customize the fill colors of rated and un-rated icons in the rating component using the `cssClass` property and the `linear-gradient` color-stops in the `background` CSS property of `.e-rating-icon`. The **first** color-stop defines the rated fill color and the **second** defines the un-rated fill color.

INDEX.JSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
  return (
    <div className='wrap'>
      <RatingComponent id='rating' value={3} cssClass='custom-
fill'></RatingComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3} cssClass='custom-fill'
        ></RatingComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

STYLES.CSS

```
/* Represents the styles for loader */
#loader {
    color: #008cff;
    font-family: 'Helvetica Neue', 'calibiri';
    font-size: 14px;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.wrap {
    margin: 50px auto;
    text-align: center;
}
.e-rating-container.custom-fill .e-rating-item-list:hover .e-rating-item-
container .e-rating-icon,
.e-rating-container.custom-fill .e-rating-item-container .e-rating-icon {
    /* To change rated symbol fill color and un-rated symbol fill color */
    background: linear-gradient(to right, #ffe814 var(--rating-value),
    #d8d7d4 var(--rating-value));
    background-clip: text;
    -webkit-background-clip: text;
}
```

This will customize the rated fill color to #ffe814 and un-rated fill color to #d8d7d4. --rating-value in the linear-gradient provides the current value of the rating item.

Changing the item spacing

You can change the space between the rating items in rating component, by using the cssClass property and setting the margin / padding CSS property of .e-rating-item-container to your desired size.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
```

```

    return (<div className='wrap'>
      <RatingComponent id='rating' value={3} cssClass='custom-
font'></RatingComponent>
    </div>);
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById('element'));

```

INDEX.TSX

```

// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

  return (
    <div className='wrap'>
      <RatingComponent id='rating' value={3} cssClass='custom-font'
></RatingComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));

```

STYLES.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  font-family: 'Helvetica Neue', 'calibiri';
  font-size: 14px;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.wrap {
  margin: 50px auto;
  text-align: center;
}
.e-rating-container.custom-font .e-rating-item-container {
  /* To change the size between items */
  margin: 0px 7px;
}

```

Changing icon using CssClass

You can change the rating item icon in rating component, you can use the `cssClass` property and set the `content` CSS property of `.e-icons.e-star-filled:before` to your desired font icon.

INDEX.JSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {
    return (<div className='wrap'>
        <RatingComponent id='rating' value={3} cssClass='custom-
icon'></RatingComponent>
        </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
// Import the Rating.
import { RatingComponent } from '@syncfusion/ej2-react-inputs';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
// To render Rating.
function App() {

    return (
        <div className='wrap'>
            <RatingComponent id='rating' value={3} cssClass='custom-icon'
></RatingComponent>
            </div>
        );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```



STYLES.CSS

```
/* Represents the styles for loader */
#loader {
    color: #008cff;
    font-family: 'Helvetica Neue', 'calibiri';
    font-size: 14px;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.wrap {
    margin: 50px auto;
    text-align: center;
}
/* Represents the styles for icon */
@font-face {
    font-family: 'custom-icon';
    src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAIAIAAwAgTlMvMjlvSfQAAAEoAAAAVmNtYXDnEudXAAABiAAAADZnbHlmVIZr
owAAAcgAAAEYaGVhZCK6KOUAAADQAAAAANmhoZWEIUAQDAAAArAAAACRobXR4CAAAAAAAYAAAAAIb
```

Accessibility in React Rating component

The Rating component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Rating component is outlined below.

Accessibility Criteria	Compatibility	
--	--	
WCAG 2.2 Support		
Section 508 Support		

```

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| Accessibility Checker Validation |  |

| Axe-core Accessibility Validation |  |

<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

```

WAI-ARIA attributes

The Rating component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Rating component:

Attributes	Purpose
<code>role=slider</code>	It defines an input where the user selects a value from within a specified range.
<code>role=button</code>	Specifies that the reset is a clickable element that resets the rating to its minimum value.
<code>aria-label</code>	Provides an accessible name for Rating.
<code>aria-valuemin</code>	It defines the minimum value of rating.
<code>aria-valuemax</code>	It defines the maximum value of rating.

| `aria-valuenow` | It defines the current value of rating. |

| `aria-hidden` | It specifies whether the reset button is interactive or not. |

Keyboard interaction

The Rating component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Rating component.

| Keyboard shortcuts | Actions |

|-----|-----|

| Space | When **Reset Button** is focused, resets to min value. |

| Arrow Up | Increases the value. |

| Arrow Left | Decreases the value; in RTL mode, increases the value. |

| Arrow Down | Decreases the value. |

| Arrow Right | Increases the value; in RTL mode, decreases the value. |

Ensuring accessibility

The Rating component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Rating component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Rating component with accessibility tools.

See also

- [Accessibility in Syncfusion React components](#)

Ribbon

Getting Started

This section explains how to create a simple Ribbon, and configure its available functionalities in React.

Dependencies

The following list of dependencies are required to use the Ribbon component in your application.

```
`javascript
```

```
|-- @syncfusion/ej2-react-ribbon
```

```
|-- @syncfusion/ej2-react-base
```

```
|-- @syncfusion/ej2-base
```

```
|-- @syncfusion/ej2-data
```

```
|-- @syncfusion/ej2-buttons
```

```
|-- @syncfusion/ej2-popups
```

```
|-- @syncfusion/ej2-splitbuttons
```

```
|-- @syncfusion/ej2-inputs
```

```
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-ribbon
\
```

Setup your development environment

To set-up a React application, choose any of the following ways. The best and easiest way is to use the [create-react-app](#). It sets up your development environment in JavaScript and improvise your application for production. Refer to the [installation instructions](#) of `create-react-app`.

```
`bash
npx create-react-app my-app
cd my-app
npm start
\
```

or

```
`bash
yarn create react-app my-app
cd my-app
yarn start
\
```

To set-up a React application in `TypeScript` environment, run the following command.

```
`bash
npx create-react-app my-app --template typescript
cd my-app
npm start
\
```

Besides using the [npm](#) package runner tool, also create an application from the `npm init`. To begin with the `npm init`, upgrade the `npm` version to `npm 6+`.

```
`bash
npm init react-app my-app
cd my-app
npm start
\
```

Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) public registry.

To install **Ribbon** component, use the following command

```
`bash
npm install @syncfusion/ej2-react-ribbon --save
`
```

The above command installs [Ribbon dependencies](#) which are required to render the component in the **React** environment.

Adding Style sheet to the Application

Add Ribbon component's styles as given below in **App.css**.

```
`css
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-ribbon/styles/material.css";
`
```

Add Ribbon to the project

Now, you can create **Ribbon** component in the application. Add **Ribbon** component in **src/App.tsx** file using the following code snippet.

```
{% raw %}
`ts
import { RibbonComponent } from "@syncfusion/ej2-react-ribbon";
import * as React from "react";
import * as ReactDOM from "react-dom";
import "../App.css";
function App() {
  return (
    <RibbonComponent id="ribbon"></RibbonComponent>
  );
}
```

```

}

const root = ReactDOM.createRoot(document.getElementById("element"));
root.render(<App />);
`

```

```
{% endraw %}
```

Injecting required modules

Inject the Ribbon required modules in your `src/App.tsx` file using the following code snippet.

```

{% raw %}
`ts

import { RibbonComponent, RibbonFileMenu, Inject, RibbonColorPicker } from "@syncfusion/ej2-react-ribbon";

import * as React from "react";
import * as ReactDOM from "react-dom";
import "./App.css";

function App() {
  return (
    <RibbonComponent id="ribbon">
      <Inject services={[RibbonFileMenu, RibbonColorPicker]} />
    </RibbonComponent>
  );
}

const root = ReactDOM.createRoot(document.getElementById("element"));
root.render(<App />);
`

{% endraw %}

```

Adding Ribbon Tab

In Ribbon, the options are arranged in tabs for easy access. You can use the `RibbonTabDirective` to define the ribbon tab like below.

```

{% raw %}
`ts

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective } from "@syncfusion/ej2-react-ribbon";

import * as React from "react";
import * as ReactDOM from "react-dom";
import "./App.css";

```

```
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home"></RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}

const root = ReactDOM.createRoot(document.getElementById("element"));
root.render(<App />);
`
```

```
{% endraw %}
```

Adding Ribbon Group

To define a ribbon group under each tab, you can use the `RibbonGroupDirective` like below. The `orientation` property of ribbon group defines whether the collection of items will be rendered column-wise or row-wise.

```
{% raw %}
```

```
`ts
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonGroupsDirective,
RibbonGroupDirective } from "@syncfusion/ej2-react-ribbon";

import * as React from "react";
import * as ReactDOM from "react-dom";

import "./App.css";

function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard" orientation="Row"></RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
```

```

</RibbonComponent>
);
}
const root = ReactDOM.createRoot(document.getElementById("element"));
root.render(<App />);
`
{% endraw %}

```

Adding Ribbon Item

You can use the `RibbonCollectionDirective` to define each ribbon collection that contains one or more items. To define each ribbon item, you can use the `RibbonItemDirective` and the `type` property to specify the type of component to be rendered, like a button, a drop-down button, a combo box, and more.

```

{% raw %}
`ts
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonCollectionsDirective, RibbonCollectionDirective, RibbonItemsDirective,
RibbonItemDirective, RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
import * as React from "react";
import * as ReactDOM from "react-dom";
import "./App.css";
function App() {
return (
<RibbonComponent id="ribbon">
<RibbonTabsDirective>
<RibbonTabDirective header="Home">
<RibbonGroupsDirective>
<RibbonGroupDirective header="Clipboard" orientation="Row">
<RibbonCollectionsDirective>
<RibbonCollectionDirective id="paste-collection">
<RibbonItemsDirective>
<RibbonItemDirective type="SplitButton" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss: "e-icons e-paste", items: [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }], content: "Paste" }}>
</RibbonItemDirective>
</RibbonItemsDirective>

```

```

</RibbonCollectionDirective>
<RibbonCollectionDirective id="cutcopy-collection">
  <RibbonItemsDirective>
    <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
  </RibbonItemDirective>
    <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
  </RibbonItemDirective>
  </RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}

const root = ReactDOM.createRoot(document.getElementById("element"));
root.render(<App />);
`

{% endraw %}

```

Run the application

Run the application in the browser using the following command:

```
npm start
```

The following example illustrates how tabs, groups, collections, and items are used in a ribbon component to form the ribbon layout.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonColorPicker, DisplayMode } from "@syncfusion/ej2-react-ribbon";

```

```

import { RibbonFileMenu, RibbonItemSize, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
    const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge format" }, { text: "Keep text only" }];
    const tableOptions = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
    const fontSize = ["8", "9", "10", "11", "12", "14", "16", "18", "20", "22", "24", "26", "28", "36", "48", "72", "96"];
    const fontStyle = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"];
    function template(props) {
        const cssClass = "ribbonTemplate " + props.activeSize;
        return (<span className={cssClass}><span className="e-icons e-video"></span><span className="text">Video</span></span>)
    }
    const fileOptions = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
        { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
        { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
        { text: "Save as", iconCss: "e-icons e-save", id: "save" }]
    return (
        <RibbonComponent id="ribbon" fileMenu={{ visible: true, menuItems: fileOptions }}>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste" showLauncherIcon={true}>
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss: "e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        <RibbonGroupDirective header="File"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        <RibbonGroupDirective header="Format"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format Painter" }}>
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format Painter" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        </RibbonComponent>
    )
}

```



```

        <RibbonGroupDirective header="Font" groupIconCss="e-
icons e-bold" isCollapsible={false} enableGroupOverflow={true}
orientation="Row" cssClass="font-group">
            <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                        <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 3, width: "150px",
allowFiltering: true }}>
                            </RibbonItemDirective>
                        <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize, index: 3, width: "65px",
allowFiltering: true }}>
                            </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
                <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                        <RibbonItemDirective
type="ColorPicker" allowedSizes={RibbonItemSize.Small}
displayOptions={DisplayMode.Simplified} colorPickerSettings={{ value:
"#123456" }}>
                            </RibbonItemDirective>
                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
bold", content: "Bold" }}>
                            </RibbonItemDirective>
                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
italic", content: "Italic" }}>
                            </RibbonItemDirective>
                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
underline", content: "Underline" }}>
                            </RibbonItemDirective>
                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
strikethrough", content: "Strikethrough" }}>
                            </RibbonItemDirective>
                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
change-case", content: "Change Case" }}>
                            </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        <RibbonGroupDirective header="Editor"
isCollapsible={false}>
            <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Large} buttonSettings={{ iconCss: "e-icons e-
edit", content: "Editor" }}>
                            </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
        </RibbonGroupDirective>
    </Ribbon>

```

```

        </RibbonCollectionDirective>
    </RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
<RibbonTabDirective header="Insert">
    <RibbonGroupsDirective>
        <RibbonGroupDirective header="Tables"
isCollapsible={false}>
            <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                        <RibbonItemDirective type="DropDown"
allowedSizes={RibbonItemSize.Large} dropDownSettings={{ iconCss: "e-icons e-
table", items: tableOptions, content: "Table" }}>
                            </RibbonItemDirective>
                        </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
            </RibbonGroupDirective>
            <RibbonGroupDirective header="Illustration"
id="illustration" groupIconCss="e-icons e-image" enableGroupOverflow={true}
showLauncherIcon={true} orientation="Row">
                <RibbonCollectionsDirective>
                    <RibbonCollectionDirective>
                        <RibbonItemsDirective>
                            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-chart", content: "Charts" }}>
                                </RibbonItemDirective>
                            </RibbonItemsDirective>
                        </RibbonCollectionDirective>
                    </RibbonCollectionsDirective>
                </RibbonGroupDirective>
                <RibbonGroupDirective header="Media"
isCollapsible={false}>
                    <RibbonCollectionsDirective>
                        <RibbonCollectionDirective>
                            <RibbonItemsDirective>
                                <RibbonItemDirective type="Template"
itemTemplate={template}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                </RibbonGroupsDirective>
            </RibbonTabDirective>
            <RibbonTabDirective header="View">
                <RibbonGroupsDirective>
                    <RibbonGroupDirective header="Views" groupIconCss="e-
icons e-print" orientation="Row">
                        <RibbonCollectionsDirective>
                            <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-print", content: "Print Layout" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonGroupsDirective>
        </RibbonTabDirective>
    </RibbonGroupsDirective>
</RibbonTabDirective>

```

```

                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-web-layout", content: "Web Layout" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Show"
isCollapsible={true}>
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective type="CheckBox"
checkBoxSettings={{ label: "Ruler", checked: false }}>
                                            </RibbonItemDirective>
                                            <RibbonItemDirective type="CheckBox"
checkBoxSettings={{ label: "Gridlines", checked: false }}>
                                            </RibbonItemDirective>
                                            <RibbonItemDirective type="CheckBox"
checkBoxSettings={{ label: "Navigation Pane", checked: true }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonFileMenu, RibbonColorPicker]} />
                                </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonColorPicker, DisplayMode } from "@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, RibbonItemSize, Inject } from "@syncfusion/ej2-
react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
import { MenuItemModel } from "@syncfusion/ej2-react-navigations";
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
    const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text:
"This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
    const fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
"20", "22", "24", "26", "28", "36", "48", "72", "96"];

```

```

const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];

function template(props: any) {
    const cssClass = "ribbonTemplate " + props.activeSize;
    return (<span className={cssClass}><span className="e-icons e-
video"></span><span className="text">Video</span></span>)
}
const fileOptions: MenuItemModel[] = [
{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
{ text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
{ text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
{ text: "Save as", iconCss: "e-icons e-save", id: "save" }]
return (
    <RibbonComponent id="ribbon" fileMenu={{ visible: true, menuItems:
fileOptions }}>
        <RibbonTabsDirective>
            <RibbonTabDirective header="Home">
                <RibbonGroupsDirective>
                    <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste" showLauncherIcon={true}>
                        <RibbonCollectionsDirective>
                            <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                    <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                        </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    <RibbonGroupDirective header="Font" groupIconCss="e-
icons e-bold" isCollapsible={false} enableGroupOverflow={true}
orientation="Row" cssClass="font-group">
                        <RibbonCollectionsDirective>
                            <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                    <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 3, width: "150px",
allowFiltering: true }}>

```

```

                                </RibbonItemDirective>
                                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize, index: 3, width: "65px",
allowFiltering: true }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="ColorPicker" allowedSizes={RibbonItemSize.Small}
displayOptions={DisplayMode.Simplified} colorPickerSettings={{ value:
"#123456" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
bold", content: "Bold" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
italic", content: "Italic" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
underline", content: "Underline" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
strikethrough", content: "Strikethrough" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
change-case", content: "Change Case" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Editor"
isCollapsible={false}>
                                    <RibbonCollectionsDirective>
                                        <RibbonCollectionDirective>
                                            <RibbonItemsDirective>
                                                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Large} buttonSettings={{ iconCss: "e-icons e-
edit", content: "Editor" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                <RibbonTabDirective header="Insert">
                                    <RibbonGroupsDirective>
                                        <RibbonGroupDirective header="Tables"
isCollapsible={false}>

```

```

        <RibbonCollectionsDirective>
            <RibbonCollectionDirective>
                <RibbonItemsDirective>
                    <RibbonItemDirective type="DropDown"
allowedSizes={RibbonItemSize.Large} dropDownSettings={{ iconCss: "e-icons e-
table", items: tableOptions, content: "Table" }}>
                        </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        <RibbonGroupDirective header="Illustration"
id="illustration" groupIconCss="e-icons e-image" enableGroupOverflow={true}
showLauncherIcon={true} orientation="Row">
            <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-chart", content: "Charts" }}>
                            </RibbonItemDirective>
                        </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
            </RibbonGroupDirective>
            <RibbonGroupDirective header="Media"
isCollapsible={false}>
                <RibbonCollectionsDirective>
                    <RibbonCollectionDirective>
                        <RibbonItemsDirective>
                            <RibbonItemDirective type="Template"
itemTemplate={template}>
                                </RibbonItemDirective>
                            </RibbonItemsDirective>
                        </RibbonCollectionDirective>
                    </RibbonCollectionsDirective>
                </RibbonGroupDirective>
            </RibbonGroupsDirective>
        </RibbonTabDirective>
        <RibbonTabDirective header="View">
            <RibbonGroupsDirective>
                <RibbonGroupDirective header="Views" groupIconCss="e-
icons e-print" orientation="Row">
                    <RibbonCollectionsDirective>
                        <RibbonCollectionDirective>
                            <RibbonItemsDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-print", content: "Print Layout" }}>
                                    </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-web-layout", content: "Web Layout" }}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                    <RibbonGroupDirective header="Show"
isCollapsible={true}>

```

```

        <RibbonCollectionsDirective>
            <RibbonCollectionDirective>
                <RibbonItemsDirective>
                    <RibbonItemDirective type="CheckBox"
checkBoxSettings={{ label: "Ruler", checked: false }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="CheckBox"
checkBoxSettings={{ label: "Gridlines", checked: false }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="CheckBox"
checkBoxSettings={{ label: "Navigation Pane", checked: true }}>
                    </RibbonItemDirective>
                </RibbonItemsDirective>
            </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
    </RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
    <Inject services={[RibbonFileMenu, RibbonColorPicker]} />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% enddraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.ribbonTemplate {
    display: flex;
    align-items: center;
    justify-content: center;
    cursor: pointer;
}
.ribbonTemplate.Large {
    flex-direction: column;
}
.ribbonTemplate.Large .e-icons {
    font-size: 35px;
}
.ribbonTemplate.Medium .e-icons,
.ribbonTemplate.Small .e-icons {
    font-size: 20px;
    margin: 15px 5px;
}
.ribbonTemplate.Small .text {

```

```

display:none;
}
/* Represents the styles for Ribbon group */
.font-group .e-ribbon-group-content {
  justify-content: center;
}

```

Modules in Ribbon component

The following modules are available in Ribbon. If the module injection type is **selective**, manual injection is required to extend the Ribbon's functionality.

Module	Description	Module Injection Type
----- ----- -----		
RibbonButton	To use the built-in button as a ribbon item.	default
RibbonCheckBox	To use the built-in checkbox as a ribbon item.	default
RibbonDropDown	To use the built-in dropdown button as a ribbon item.	default
RibbonSplitButton	To use the built-in split button as a ribbon item.	default
RibbonComboBox	To use the built-in combobox as a ribbon item.	default
RibbonGroupButton	To use the built-in groupbutton as a ribbon item.	default
RibbonColorPicker	Inject this module to use the built-in colorpicker as a ribbon item.	selective
RibbonGallery	Inject this module to use the gallery as a ribbon item.	selective
RibbonFileMenu	Inject this module to use the file menu feature.	selective
RibbonBackstage	Inject this module to use the backstage view feature.	selective
RibbonContextualTab	Inject this module to use the contextual tab feature.	selective
RibbonKeyTip	Inject this module to use the keytip feature.	selective

These modules should be injected into the Ribbon using the **Inject** directive.

INDEX.JSX

```

{% raw %}
import { RibbonFileMenu, RibbonColorPicker, Inject } from "@syncfusion/ej2-react-ribbon";
<RibbonComponent id="ribbon" fileMenu={{ visible: true, menuItems:
fileOptions}}>
  // Render Tabs here
<Inject services={[RibbonFileMenu, RibbonColorPicker]} />
</RibbonComponent>
{% endraw %}

```

Tabs and Groups

The Ribbon component consists of a series of tabs that are organized into groups to enable quick access to specific commands or tools. Each tab contains a set of groups, and each group contains collections of items that are logically related to each other.

Adding Tabs

You can use the [tabs](#) property to add tabs to the Ribbon component and define the content of the tab header by using the [header](#) property.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
        </RibbonTabDirective>
        <RibbonTabDirective header="Insert">
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
        </RibbonTabDirective>
        <RibbonTabDirective header="Insert">
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}
```

INDEX.CSS

```
/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
```

```

left: 45%;
position: absolute;
top: 45%;
width: 30%;
}

```

Adding Groups

You can use the [groups](#) property to add groups for each tab in the Ribbon and define the name of the group header by using the [header](#) property.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonGroupsDirective, RibbonGroupDirective } from "@syncfusion/ej2-react-
ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
        <RibbonTabDirective header="Insert">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Tables">
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonGroupsDirective, RibbonGroupDirective } from "@syncfusion/ej2-react-
ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">

```

```

        <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
            </RibbonGroupDirective>
        </RibbonGroupsDirective>
    </RibbonTabDirective>
    <RibbonTabDirective header="Insert">
        <RibbonGroupsDirective>
            <RibbonGroupDirective header="Tables">
            </RibbonGroupDirective>
        </RibbonGroupsDirective>
    </RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>

    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

Adding Items

You can add collections of items to each group by using the [collections](#) and [items](#) properties.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
    const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>

```

```

                                <RibbonItemDirective
type="SplitButton"
                                splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>

```

```

                                <RibbonItemDirective
type="SplitButton"
                                splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

For more information on the built-in and how to add custom Ribbon items, you can visit the [items](#) page.

Ribbon Items

Ribbon renders various built-in items by using the `RibbonItemDirective` tag, to specify the [type](#) property. By default, the type property is set as `Button` which renders the Button.

Built-in items

You can render the built-in Ribbon items by using the [items](#) property, to specify the [type](#) property.

The following table explains the built-in items and their actions.

| Built-in Ribbon Items | Actions |

|-----|-----|

| Button | Renders button as ribbon item.|

| CheckBox | Renders checkbox as ribbon item.|

| DropDown | Renders dropdownbutton as ribbon item.|

| SplitButton | Renders splitbutton as ribbon item.|

| ComboBox | Renders combobox as ribbon item.|

| ColorPicker | Renders color picker as ribbon item.|

| GroupButton | Renders groupbutton as ribbon item.|

Button items

You can render the built-in button Ribbon item by setting the [type](#) property as `Button`. You can also customize the button item using the [RibbonButtonSettings](#), which provides options such as `IconCss`, `Content`, `IsToggle` and more.

Toggle button

The [isToggle](#) property can be used to define whether the button act as a toggle button or not. By default, the value is `false`.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut", isToggle: true
}}}
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
```

```

}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut", isToggle: true
}}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Checkbox items

You can render the built-in checkBox Ribbon item by setting the [type](#) property to `CheckBox`. You can also customize the checkBox item using the [RibbonCheckBoxSettings](#), which provides options such as `LabelPosition`, `Label`, `Checked` and more.

Checkbox state

You can use the [checked](#) property is used to handle the checked or unchecked state. By default, the value is `false`.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="CheckBox"
checkBoxSettings={{ checked: true, label:"Ruler" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  return (

```



```

    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="CheckBox"
checkBoxSettings={{ checked: true, label: "Ruler" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Defining label

You can use the [label](#) property to add a caption for the CheckBox. The label position can be set **Before** or **After**, by using the [labelPosition](#) property. By default, the labelPosition is **After**.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">

```

```

        <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
                <RibbonCollectionsDirective>
                    <RibbonCollectionDirective>
                        <RibbonItemsDirective>
                            <RibbonItemDirective type="CheckBox"
checkBoxSettings={{ checked: true, label: "Ruler", labelPosition: "Before"
}}>
                        </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
        </RibbonGroupDirective>
    </RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="CheckBox"
checkBoxSettings={{ checked: true, label: "Ruler", labelPosition: "Before"
}}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                </RibbonGroupsDirective>
            </RibbonTabDirective>
        </RibbonTabsDirective>
    </RibbonComponent>
    );
}

```

```
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}
```

INDEX.CSS

```
/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
```

DropDown button items

You can render the built-in dropDown Ribbon item by setting the [type](#) property to `DropDown`. You can also customize the dropDown item through [RibbonDropDownSettingsModel](#), which provides options such as `iconCss`, `content`, `target` and more.

Target

The [target](#) property specifies the element selector to be displayed in the DropDownButton popup.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { ListViewComponent } from '@syncfusion/ej2-react-lists';
function App() {
  return (
    <div>
      <RibbonComponent id="ribbon">
        <RibbonTabsDirective>
          <RibbonTabDirective header="Home">
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="DropDown" dropDownSettings={{ iconCss: "e-icons e-image", content:
"Pictures", target: "#pictureList" }}>
                        </RibbonItemDirective>
                      </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                  </RibbonCollectionsDirective>
                </RibbonGroupDirective>
              </RibbonGroupsDirective>
            </RibbonTabDirective>
          </RibbonTabsDirective>
        </RibbonComponent>
      </div>
    );
  }
}
```

```

        </RibbonTabsDirective>
    </RibbonComponent>
    <ListViewComponent id='pictureList' dataSource={['This device',
    'Stock Images', 'Online Images']} showHeader={true} headerTitle="Insert
    Picture From"></ListViewComponent>
</div>

);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { ListViewComponent } from '@syncfusion/ej2-react-lists';
function App() {
    return (
        <div>
            <RibbonComponent id="ribbon">
                <RibbonTabsDirective>
                    <RibbonTabDirective header="Home">
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Clipboard">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="DropDown" dropDownSettings={{ iconCss: "e-icons e-image", content:
"Pictures", target: "#pictureList" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                            </RibbonGroupsDirective>
                        </RibbonTabDirective>
                    </RibbonTabsDirective>
                </RibbonComponent>
                <ListViewComponent id='pictureList' dataSource={['This device',
'Stock Images', 'Online Images']} showHeader={true} headerTitle="Insert
Picture From"></ListViewComponent>
            </div>
        );
    }
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Customize Dropdown button item

You can customize the dropdown button item by specifying a custom cssClass using the [beforeItemRender](#) event.

The following sample showcases how to customize a specific dropdown item.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  const tableOptions = [{ text: "Insert Table" }, { text: "This device" },
{ text: "Convert Table" }, { text: "Excel Spreadsheet" }];
  return (
    <div>
      <RibbonComponent id="ribbon">
        <RibbonTabsDirective>
          <RibbonTabDirective header="Insert">
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Tables">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="DropDown" dropDownSettings={{ iconCss: "e-icons e-table", items:
tableOptions, content: "Table", beforeItemRender: function (args) {
                                if (args.item.text ===
'Insert Table') {
args.element.classList.add("e-custom-class");
                                }
                                } }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </div>
  );
}

```

```

        </RibbonComponent>
    </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { ItemModel, MenuEventArgs } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text:
"This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
    return (
        <div>
            <RibbonComponent id="ribbon">
                <RibbonTabsDirective>
                    <RibbonTabDirective header="Insert">
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Tables">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="DropDown" dropDownSettings={{ iconCss: "e-icons e-table", items:
tableOptions, content: "Table", beforeItemRender: function (args:
MenuEventArgs) {
                                                                    if (args.item.text ===
'Insert Table') {
args.element.classList.add("e-custom-class");
                                                                    }
                                                                    } }>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
            </RibbonComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));

```

```
{% endraw %}
```

INDEX.CSS

```
/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.e-custom-class {
  color: green;
}
```

Split button items

You can render the built-in splitButton Ribbon item by setting the `type` property to `SplitButton`. You can also customize the splitButton item through [RibbonSplitButtonSettingsModel](#), which provides options such as `iconCss`, `items`, `target` and more.

Target

The `target` property specifies the element selector to be displayed in the SplitButton popup.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { ListViewComponent } from '@syncfusion/ej2-react-lists';
function App() {
  return (
    <div>
      <RibbonComponent id="ribbon">
        <RibbonTabsDirective>
          <RibbonTabDirective header="Home">
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-image",
content: "Pictures", target: "#pictureList" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
```

```

        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
    <ListViewComponent id='pictureList' dataSource={['This device',
'Stock Images', 'Online Images']} showHeader={true} headerTitle="Insert
Picture From"></ListViewComponent>
  </div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { ListViewComponent } from '@syncfusion/ej2-react-lists';
function App() {
  return (
    <div>
      <RibbonComponent id="ribbon">
        <RibbonTabsDirective>
          <RibbonTabDirective header="Home">
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-image",
content: "Pictures", target: "#pictureList" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
      </RibbonComponent>
      <ListViewComponent id='pictureList' dataSource={['This device',
'Stock Images', 'Online Images']} showHeader={true} headerTitle="Insert
Picture From"></ListViewComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```


INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Combobox items

You can render the built-in comboBox Ribbon item by setting the `type` property to `ComboBox`. You can also customize the comboBox item through [RibbonComboBoxSettingsModel](#), which provides options such as `allowFiltering`, `autoFill`, `index`, `sortOrder` and more.

Filtering

You can use the `allowFiltering` property to filter the data items. The filtering operation is initiated automatically, as soon as you start typing characters. If no match is found the value of the `noRecordsTemplate` property will be displayed. By default, the value is `false`.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  const fontStyle = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria
Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe
Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, allowFiltering: true }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
  );
}

```

```

}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
    const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, allowFiltering: true }}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                </RibbonGroupsDirective>
            </RibbonTabDirective>
        </RibbonTabsDirective>
    </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

[Index](#)

You can use the [index](#) property to get or set the selected item in the combobox.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
    const fontStyle = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria
Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe
Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 2 }}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                </RibbonGroupsDirective>
            </RibbonTabDirective>
        </RibbonTabsDirective>
    </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
    const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
{% endraw %}
```

```

"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];
    return (
      <RibbonComponent id="ribbon">
        <RibbonTabsDirective>
          <RibbonTabDirective header="Home">
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 2 }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

SortOrder

You can use the [sortOrder](#) property to specify the order in which the DataSource should be sorted.

None	The data source is not sorted.
Ascending	The data source is sorted in ascending order.
Descending	The data source is sorted in descending order.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";

```

```

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
    const fontStyle = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria
Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe
Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, sortOrder: "Descending" }}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                </RibbonGroupsDirective>
            </RibbonTabDirective>
        </RibbonTabsDirective>
    </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
    const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>

```

```

                                <RibbonItemsDirective>
                                    <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, sortOrder: "Descending" }}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                </RibbonGroupsDirective>
            </RibbonTabDirective>
        </RibbonTabsDirective>
    </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

Colorpicker items

You can render the built-in colorPicker Ribbon item by setting the [type](#) property to `ColorPicker`. You can also customize the colorPicker item through [RibbonColorPickerSettingsModel](#), which provides options such as `value`, `columns`, `showButtons` and more.

Value

You can use the [value](#) property to specify the color value. The value should be specified as Hex code.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonColorPicker, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>

```

```

                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="ColorPicker" colorPickerSettings={{ value: "035a" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        <Inject services={[RibbonColorPicker]}></Inject>
    </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonColorPicker, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="ColorPicker" colorPickerSettings={{ value: "035a" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        <Inject services={[RibbonColorPicker]}></Inject>
    </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Groupbutton items

You can render the built-in groupbutton Ribbon item by setting the [type](#) property to `GroupButton`. You can also customize the groupbutton item using the `RibbonGroupButtonSettingsModel`, which provides options such as `selection` and `items`.

Items

You can render the groupbutton items by using [items](#) property. You can also customize the groupbutton items through [RibbonGroupButtonItemModel](#), which provides options such as [content](#), [iconCss](#), [selected](#) and more.

Item content

You can use the [content](#) property to define the text content for the groupbutton.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  const groupButtonItem: RibbonGroupButtonSettingsModel = {
```

```
    items: [
```

```
      { iconCss: 'e-icons e-align-left', content: 'Align Left' },
```

```
      { iconCss: 'e-icons e-align-center',content: 'Align Center' },
```

```
      { iconCss: 'e-icons e-align-right',content: 'Align Right' },
```

```
      { iconCss: 'e-icons e-justify',content: 'Justify' }
    ]
  }
}
```

```
return (
```



```

<RibbonComponent id="ribbon">
  <RibbonTabsDirective>
    <RibbonTabDirective header="Home" >
      <RibbonGroupsDirective>
        <RibbonGroupDirective header="Paragraph">
          <RibbonCollectionsDirective>
            <RibbonCollectionDirective>
              <RibbonItemsDirective>
                <RibbonItemDirective type="GroupButton" allowedSizes={RibbonItemSize.Medium}
groupButtonSettings={ groupButtonItem }>
              </RibbonItemDirective>
            </RibbonItemsDirective>
          </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
      </RibbonGroupDirective>
    </RibbonGroupsDirective>
  </RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}

export default App;

ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

Icon only

You can use the [iconCss](#) property to customize the groupbutton icon. If the `iconCss` property is not defined, the groupbutton will not be rendered.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {

```

```

const groupButtonItem = {
  items: [
    { iconCss: 'e-icons e-align-left' },
    { iconCss: 'e-icons e-align-center' },
    { iconCss: 'e-icons e-align-right' },
    { iconCss: 'e-icons e-justify' }
  ]
}
return (
  <RibbonComponent id="ribbon">
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home" >
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Paragraph">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="GroupButton"
allowedSizes={RibbonItemSize.Medium} groupButtonSettings={ groupButtonItem }>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize, RibbonGroupButtonSettingsModel } from "@syncfusion/ej2-react-
ribbon";
function App() {

  const groupButtonItem: RibbonGroupButtonSettingsModel = {
    items: [
      { iconCss: 'e-icons e-align-left' },
      { iconCss: 'e-icons e-align-center' },
      { iconCss: 'e-icons e-align-right' },
      { iconCss: 'e-icons e-justify' }
    ]
  }
  return (

```

```

    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Paragraph">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="GroupButton"
allowedSizes={RibbonItemSize.Medium} groupButtonSettings={ groupButtonItem }>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
      </RibbonComponent>
    );
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById("element"));
  {% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Selection

You can use the [selected](#) property to select the groupbutton item initially. When set to `true`, the button will be selected. By default the `selected` property is false.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective ,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {

  const groupButtonItem = {
    items: [

```

```

    { iconCss: 'e-icons e-align-left', selected: true, content: 'Align
Left' },
    { iconCss: 'e-icons e-align-center',content: 'Align Center' },
    { iconCss: 'e-icons e-align-right',content: 'Align Right' },
    { iconCss: 'e-icons e-justify',content: 'Justify' }
  ]
}
return (
  <RibbonComponent id="ribbon">
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home" >
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Paragraph">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="GroupButton"
allowedSizes={RibbonItemSize.Medium} groupButtonSettings={ groupButtonItem }>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize, RibbonGroupButtonSettingsModel } from "@syncfusion/ej2-react-
ribbon";
function App() {

  const groupButtonItem: RibbonGroupButtonSettingsModel = {
    items: [
      { iconCss: 'e-icons e-align-left', selected: true, content: 'Align
Left' },
      { iconCss: 'e-icons e-align-center',content: 'Align Center' },
      { iconCss: 'e-icons e-align-right',content: 'Align Right' },
      { iconCss: 'e-icons e-justify',content: 'Justify' }
    ]
  }

  return (
    <RibbonComponent id="ribbon">

```

```

        <RibbonTabsDirective>
            <RibbonTabDirective header="Home" >
                <RibbonGroupsDirective>
                    <RibbonGroupDirective header="Paragraph">
                        <RibbonCollectionsDirective>
                            <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                    <RibbonItemDirective type="GroupButton"
allowedSizes={RibbonItemSize.Medium} groupButtonSettings={ groupButtonItem }>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

Single selection

You can set the [selection](#) property value as `RibbonGroupButtonSelection.Single` to make one selection at a time. It automatically deselects the previous choice when a different item is clicked.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective ,
RibbonGroupButtonSelection, RibbonItemSize } from "@syncfusion/ej2-react-
ribbon";
function App() {

    const groupButtonItem = {
        selection: RibbonGroupButtonSelection.Single,
        items: [

```

```

    { iconCss: 'e-icons e-align-left', selected: true, content: 'Align
Left' },
    { iconCss: 'e-icons e-align-center',content: 'Align Center' },
    { iconCss: 'e-icons e-align-right',content: 'Align Right' },
    { iconCss: 'e-icons e-justify',content: 'Justify' }
  ]
}
return (
  <RibbonComponent id="ribbon">
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home" >
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Paragraph">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="GroupButton"
allowedSizes={RibbonItemSize.Medium} groupButtonSettings={ groupButtonItem }>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonGroupButtonSelection, RibbonItemSize, RibbonGroupButtonSettingsModel }
from "@syncfusion/ej2-react-ribbon";
function App() {

  const groupButtonItem: RibbonGroupButtonSettingsModel = {
    selection: RibbonGroupButtonSelection.Single,
    items: [
      { iconCss: 'e-icons e-align-left', selected: true, content: 'Align
Left' },
      { iconCss: 'e-icons e-align-center',content: 'Align Center' },
      { iconCss: 'e-icons e-align-right',content: 'Align Right' },
      { iconCss: 'e-icons e-justify',content: 'Justify' }
    ]
  }
  return (

```

```

    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Paragraph">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="GroupButton"
allowedSizes={RibbonItemSize.Medium} groupButtonSettings={ groupButtonItem }>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
      </RibbonComponent>
    );
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById("element"));
  {% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Multiple selection

You can set the [selection](#) property value as `RibbonGroupButtonSelection.Multiple` to select more than one button at a time. Users can select a button one by one to select multiple buttons.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective ,
RibbonGroupButtonSelection, RibbonItemSize } from "@syncfusion/ej2-react-
ribbon";
function App() {

  const groupButtonItem = {
    selection: RibbonGroupButtonSelection.Multiple,

```

```

    items: [
      { iconCss: 'e-icons e-bold', content: 'Bold', selected: true },
      { iconCss: 'e-icons e-italic', content: 'Italic' },
      { iconCss: 'e-icons e-underline', content: 'Underline' },
      { iconCss: 'e-icons e-strikethrough', content: 'Strikethrough' },
      { iconCss: 'e-icons e-change-case', content: 'Change Case' }
    ]
  }
}
return (
  <RibbonComponent id="ribbon">
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home" >
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Paragraph">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="GroupButton"
allowedSizes={RibbonItemSize.Medium} groupButtonSettings={ groupButtonItem }>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonGroupButtonSelection, RibbonItemSize, RibbonGroupButtonSettingsModel }
from "@syncfusion/ej2-react-ribbon";
function App() {

  const groupButtonItem: RibbonGroupButtonSettingsModel = {
    selection: RibbonGroupButtonSelection.Multiple,
    items: [
      { iconCss: 'e-icons e-bold', content: 'Bold', selected: true },
      { iconCss: 'e-icons e-italic', content: 'Italic' },
      { iconCss: 'e-icons e-underline', content: 'Underline' },
      { iconCss: 'e-icons e-strikethrough', content: 'Strikethrough' },
      { iconCss: 'e-icons e-change-case', content: 'Change Case' }
    ]
  }
}

```



```

return (
  <RibbonComponent id="ribbon">
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home" >
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Paragraph">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="GroupButton"
allowedSizes={RibbonItemSize.Medium} groupButtonSettings={ groupButtonItem }>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Groupbutton in simplified mode layout

In simplified mode, the groupbutton will be rendered as a dropdownbutton. The dropdownbutton icon will be updated based on the button item selected. The initial button icon will be the set, if none of the buttons are selected.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective ,
RibbonGroupButtonSelection, RibbonItemSize } from "@syncfusion/ej2-react-
ribbon";
function App() {

```

```

const groupButtonItem = {
  selection: RibbonGroupButtonSelection.Single,
  items: [
    { iconCss: 'e-icons e-align-left', selected: true, content: 'Align
Left' },
    { iconCss: 'e-icons e-align-center', content: 'Align Center' },
    { iconCss: 'e-icons e-align-right', content: 'Align Right' },
    { iconCss: 'e-icons e-justify', content: 'Justify' }
  ]
}
return (
  <RibbonComponent id="ribbon" activeLayout='Simplified'>
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home" >
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Paragraph">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="GroupButton"
allowedSizes={RibbonItemSize.Medium} groupButtonSettings={ groupButtonItem }>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonGroupButtonSelection, RibbonItemSize, RibbonGroupButtonSettingsModel }
from "@syncfusion/ej2-react-ribbon";
function App() {

  const groupButtonItem: RibbonGroupButtonSettingsModel = {
    selection: RibbonGroupButtonSelection.Single,
    items: [
      { iconCss: 'e-icons e-align-left', selected: true, content: 'Align
Left' },
      { iconCss: 'e-icons e-align-center', content: 'Align Center' },
      { iconCss: 'e-icons e-align-right', content: 'Align Right' },
      { iconCss: 'e-icons e-justify', content: 'Justify' }
    ]
  }
}

```

```

    ]
  }
  return (
    <RibbonComponent id="ribbon" activeLayout='Simplified'>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Paragraph">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="GroupButton"
allowedSizes={RibbonItemSize.Medium} groupButtonSettings={ groupButtonItem }>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
      </RibbonComponent>
    );
  }
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Custom items

You can customize the ribbon items with non-built-in items or HTML content by setting the [type](#) property to `Template`. This provides an option to customize the ribbon items with greater flexibility.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  function template(props) {

```

```

        const cssClass = "ribbonTemplate " + props.activeSize;
        return (<span className={cssClass}><span className="e-icons e-
video"></span><span className="text">Video</span></span>)
    }
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Template"
itemTemplate={template}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
            </RibbonComponent>
        );
    }
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
    function template(props:any) {
        const cssClass = "ribbonTemplate " + props.activeSize;
        return (<span className={cssClass}><span className="e-icons e-
video"></span><span className="text">Video</span></span>)
    }
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Template"
itemTemplate={template}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
            </RibbonComponent>
        );
    }
}

```

```

        </RibbonItemDirective>
      </RibbonItemsDirective>
    </RibbonCollectionDirective>
  </RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% enddraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.ribbonTemplate {
  display: flex;
  align-items: center;
  justify-content: center;
  cursor: pointer;
}
.ribbonTemplate.Large {
  flex-direction: column;
}
.ribbonTemplate.Large .e-icons {
  font-size: 35px;
}
.ribbonTemplate.Medium .e-icons,
.ribbonTemplate.Small .e-icons {
  font-size: 20px;
  margin: 15px 5px;
}
.ribbonTemplate.Small .text {
  display: none;
}
/* Represents the styles for Ribbon group */
.font-group .e-ribbon-group-content {
  justify-content: center;
}

```

Items display Mode

You can use the [displayOptions](#) property to display the items in the Ribbon.

Auto	The items are displayed in all layouts based on the ribbon's overflow state.
Classic	The items are displayed only in the classic layout group.
Simplified	The items are displayed only in the simplified layout group.
Overflow	The items are displayed only in the overflow popup.

Display items in Classic only

To display the items only in the classic layout group, set the mode as `DisplayMode.Classic` in the [displayOptions](#) property.

```
{% raw %}
`ts
import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, DisplayMode } from "@syncfusion/ej2-react-ribbon";

function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button" displayOptions={DisplayMode.Classic} buttonSettings={{ iconCss:
                    "e-icons e-cut", content: "Cut" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
```

```

</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

Display items in Simplified only

To display the items only in the simplified layout group, set the mode as `DisplayMode.Simplified` in the [displayOptions](#) property.

```
{% raw %}
```

```
`ts
```

```

import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, DisplayMode } from "@syncfusion/ej2-react-ribbon";

function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button" displayOptions={DisplayMode.Simplified} buttonSettings={{
iconCss: "e-icons e-cut", content: "Cut" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}

```

```

</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

Display items in Overflow popup only

To display the items only in the overflow, set the mode as `DisplayMode.Overflow` in the [displayOptions](#) property.

```
{% raw %}
```

```
`ts
```

```

import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, DisplayMode } from "@syncfusion/ej2-react-ribbon";

function App() {
return (
<RibbonComponent id="ribbon">
<RibbonTabsDirective>
<RibbonTabDirective header="Home">
<RibbonGroupsDirective>
<RibbonGroupDirective header="Clipboard">
<RibbonCollectionsDirective>
<RibbonCollectionDirective>
<RibbonItemsDirective>
<RibbonItemDirective type="Button" displayOptions={DisplayMode.Overflow} buttonSettings={{
iconCss: "e-icons e-cut", content: "Cut" }}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>

```



```

</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}

export default App;

ReactDOM.render(<App />, document.getElementById("element"));
,

{% endraw %}

```

Enable or disable items

You can use the [disabled](#) property to disable a Ribbon item. It prevents the user interaction when set to `true`. By default, the value is `false`.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
disabled={true} buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut",
isToggle: true }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="CheckBox"
disabled={true} checkBoxSettings={{ checked: true, label: "Ruler" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="DropDown"
disabled={true} dropDownSettings={{ content: 'Table', iconCss: 'e-icons e-
table' }}>
                    </RibbonItemDirective>

```

```

                                <RibbonItemDirective
type="SplitButton" disabled={true} splitButtonSettings={{ content: 'Table',
iconCss: 'e-icons e-table' }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
disabled={true} buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut",
isToggle: true }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="CheckBox"
disabled={true} checkBoxSettings={{ checked: true, label: "Ruler" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="DropDown"
disabled={true} dropDownSettings={{ content: 'Table', iconCss: 'e-icons e-
table' }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective
type="SplitButton" disabled={true} splitButtonSettings={{ content: 'Table',
iconCss: 'e-icons e-table' }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        </RibbonComponent>
    );
}
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

```

        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Ribbon Layouts

The Ribbon allows to customize the layout by using the [activeLayout](#) property. The Ribbon component supports the following layouts:

Classic layout

In classic layout, the Ribbon component organizes the items and groups in a traditional form by setting the [activeLayout](#) property to [Classic](#). By default, the Ribbon component renders in the [Classic](#) layout.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
  const tableOptions = [{ text: "Insert Table" }, { text: "This device" },
{ text: "Convert Table" }, { text: "Excel Spreadsheet" }];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton"

```

```

splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
  </RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
<RibbonCollectionDirective>
  <RibbonItemsDirective>
    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
      </RibbonItemDirective>
    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
      </RibbonItemDirective>
    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
      </RibbonItemDirective>
    </RibbonItemsDirective>
  </RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
<RibbonTabDirective header="Insert">
  <RibbonGroupsDirective>
    <RibbonGroupDirective header="Tables">
      <RibbonCollectionsDirective>
        <RibbonCollectionDirective>
          <RibbonItemsDirective>
            <RibbonItemDirective type="DropDown"
dropDownSettings={{ iconCss: "e-icons e-table", items: tableOptions, content:
"Table" }}>
              </RibbonItemDirective>
            </RibbonItemsDirective>
          </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
      </RibbonGroupDirective>
    </RibbonGroupsDirective>
  </RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";

```

```

import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
    const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text:
"This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton"
                                            splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
                <RibbonTabDirective header="Insert">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Tables">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
                <RibbonTabDirective header="Insert">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Tables">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="DropDown"
dropDownSettings={{ iconCss: "e-icons e-table", items: tableOptions, content:
"Table" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        </RibbonComponent>
    );
}

```

```

        </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

Defining items size

You can use the [allowedSizes](#) property to set the allowed size for an item. The Ribbon items can be appeared in three different sizes: Large(large icon with text), Medium(small icon with text) and Small(small icon only). On resizing, the items size can be changed based on the available width of the tab content from the order of Large-> Medium-> Small and viceversa.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
    const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective>
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
                                        splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionDirective>
                    </RibbonGroupDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        </RibbonComponent>
    );
}
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

```

                                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
            <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
cut", content: "Cut" }}>

```

```

                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Defining items orientation

The Ribbon group [orientation](#) property allows to manage how the items are aligned either in a **Row** or **Column**. By default, the orientation is set to **Column**, in which the items are arranged vertically.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
  const fontSize = ["8", "9", "10", "11", "12", "14", "16", "18", "20",
"22", "24", "26", "28", "36", "48", "72", "96"];
  const fontStyle = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria
Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe
Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>

```



```

        <RibbonTabDirective header="Home">
            <RibbonGroupsDirective>
                <RibbonGroupDirective orientation="Column">
                    <RibbonCollectionsDirective>
                        <RibbonCollectionDirective>
                            <RibbonItemsDirective>
                                <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
                                splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                </RibbonItemDirective>
                            </RibbonItemsDirective>
                        </RibbonCollectionDirective>
                        <RibbonCollectionDirective>
                            <RibbonItemsDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                                </RibbonItemDirective>
                            </RibbonItemsDirective>
                        </RibbonCollectionDirective>
                    </RibbonCollectionsDirective>
                </RibbonGroupDirective>
                <RibbonGroupDirective orientation="Row">
                    <RibbonCollectionsDirective>
                        <RibbonCollectionDirective>
                            <RibbonItemsDirective>
                                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 2 }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize, index: 4 }}>
                                </RibbonItemDirective>
                            </RibbonItemsDirective>
                        </RibbonCollectionDirective>
                        <RibbonCollectionDirective>
                            <RibbonItemsDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-bold", content: "Bold" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-italic", content: "Italic" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-underline", content: "Underline" }}>
                                </RibbonItemDirective>
                            </RibbonItemsDirective>
                        </RibbonCollectionDirective>
                    </RibbonCollectionsDirective>
                </RibbonGroupDirective>
            </RibbonGroupsDirective>
        </RibbonTabDirective>

```

```

        </RibbonTabDirective>
    </RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
    const fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
"20", "22", "24", "26", "28", "36", "48", "72", "96"];
    const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective orientation="Column">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
            </RibbonComponent>

```

```

        </RibbonItemsDirective>
      </RibbonCollectionDirective>
    </RibbonCollectionsDirective>
  </RibbonGroupDirective>
  <RibbonGroupDirective orientation="Row">
    <RibbonCollectionsDirective>
      <RibbonCollectionDirective>
        <RibbonItemsDirective>
          <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle }}>
            </RibbonItemDirective>
          <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize }}>
            </RibbonItemDirective>
          </RibbonItemsDirective>
        </RibbonCollectionDirective>
        <RibbonCollectionDirective>
          <RibbonItemsDirective>
            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-bold", content: "Bold" }}>
              </RibbonItemDirective>
            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-italic", content: "Italic" }}>
              </RibbonItemDirective>
            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-underline", content: "Underline" }}>
              </RibbonItemDirective>
            </RibbonItemsDirective>
          </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
      </RibbonGroupDirective>
    </RibbonGroupsDirective>
  </RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% enddraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

When the orientation is set to **Row** a group may have a maximum of three collections each of which may contain any number of items. When the orientation is set to **Column** a group may have any

number of collections, each of which may contain one large-sized item or three medium/small-sized items. If two large-sized items are specified, it automatically converts into two medium/small-sized items.

Defining group header

You can use the [header](#) property to set the name for each group header.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
          <RibbonGroupDirective header="Font">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-bold", content: "Bold" }}>
                </RibbonItemDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-italic", content: "Italic" }}>
                </RibbonItemDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-underline", content: "Underline" }}>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>

```

```

        </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
                                        splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    <RibbonGroupDirective header="Font">
                        <RibbonCollectionsDirective>
                            <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-bold", content: "Bold" }}>
                                    </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-italic", content: "Italic" }}>
                                    </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-underline", content: "Underline" }}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                </RibbonGroupsDirective>
            </RibbonTabDirective>
        </RibbonTabsDirective>
    </RibbonComponent>

```

```

        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Defining group icon

You can use the [groupIconCss](#) property to customize the icons in the group overflow button. When the ribbon's size is adjusted, the group popup will appear.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
  const fontSize = ["8", "9", "10", "11", "12", "14", "16", "18", "20",
"22", "24", "26", "28", "36", "48", "72", "96"];
  const fontStyle = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria
Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe
Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}

```

```

splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
    </RibbonItemDirective>
    </RibbonItemsDirective>
    </RibbonCollectionDirective>
    <RibbonCollectionDirective>
        <RibbonItemsDirective>
            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                </RibbonItemDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                        </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        <RibbonGroupDirective header="Font" groupIconCss="e-
icons e-bold">
            <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                        <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 2, width: "150px",
allowFiltering: true }}>
                            </RibbonItemDirective>
                            <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize, index: 4, width: "65px",
allowFiltering: true }}>
                                </RibbonItemDirective>
                            </RibbonItemsDirective>
                        </RibbonCollectionDirective>
                    <RibbonCollectionDirective>
                        <RibbonItemsDirective>
                            <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
bold", content: "Bold" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
italic", content: "Italic" }}>
                                    </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
underline", content: "Underline" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>

```

```

        </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
    const fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
"20", "22", "24", "26", "28", "36", "48", "72", "96"];
    const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
                                        splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
            </RibbonComponent>
        );
    }
}
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```



```

        </RibbonCollectionDirective>
    </RibbonCollectionsDirective>
</RibbonGroupDirective>
<RibbonGroupDirective header="Font" groupIconCss="e-
icons e-bold">
    <RibbonCollectionsDirective>
        <RibbonCollectionDirective>
            <RibbonItemsDirective>
                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 2, width: "150px",
allowFiltering: true }}>
            </RibbonItemDirective>
                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize, index: 4, width: "65px",
allowFiltering: true }}>
            </RibbonItemDirective>
            </RibbonItemsDirective>
        </RibbonCollectionDirective>
        <RibbonCollectionDirective>
            <RibbonItemsDirective>
                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
bold", content: "Bold" }}>
            </RibbonItemDirective>
                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
italic", content: "Italic" }}>
            </RibbonItemDirective>
                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Small} buttonSettings={{ iconCss: "e-icons e-
underline", content: "Underline" }}>
            </RibbonItemDirective>
            </RibbonItemsDirective>
        </RibbonCollectionDirective>
    </RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

```
}

```

Enabling group launcher icon

You can use the [showLauncherIcon](#) property to enable or disable the launcher icon for each group. By default, the property is set to `false`.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
showLauncherIcon={true}>
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                      </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
```

```

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
showLauncherIcon={true}>
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                      </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Customize launcher icon

You can use the [launcherIconCss](#) property to customize the launcher icon by applying the custom styles.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon" launcherIconCss="e-icons e-description">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
showLauncherIcon={true}>
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                      </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon" launcherIconCss="e-icons e-description">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
```

```

        <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
showLauncherIcon={true}>
                <RibbonCollectionsDirective>
                    <RibbonCollectionDirective>
                        <RibbonItemsDirective>
                            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                </RibbonItemDirective>
                            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                </RibbonItemDirective>
                            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                                </RibbonItemDirective>
                        </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
            </RibbonGroupDirective>
        </RibbonGroupsDirective>
    </RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>

    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

Defining group collapsible state

You can use the [isCollapsible](#) property to determine whether the group is collapsed or not during resize. By default, the property is set to `true`. To prevent the group from being collapsed, set the property to `false`.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,

```

```

RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
    const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
    const fontSize = ["8", "9", "10", "11", "12", "14", "16", "18", "20",
"22", "24", "26", "28", "36", "48", "72", "96"];
    const fontStyle = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria
Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe
Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
                                        splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        <RibbonGroupDirective header="Font"
isCollapsible={false}>
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 2 }}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize, index: 4 }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
            </RibbonComponent>
        );
    }
}

```

```

                                <RibbonItemsDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-bold", content: "Bold" }}>
                                        </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-italic", content: "Italic" }}>
                                        </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-underline", content: "Underline" }}>
                                        </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                </RibbonGroupsDirective>
            </RibbonTabDirective>
        </RibbonTabsDirective>
    </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% enddraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
    const fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
"20", "22", "24", "26", "28", "36", "48", "72", "96"];
    const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}

```

```

                                splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Font"
isCollapsible={false}>
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-bold", content: "Bold" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-italic", content: "Italic" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-underline", content: "Underline" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                </RibbonComponent>
                                );
                                }
                                export default App;
                                ReactDOM.render(<App />, document.getElementById("element"));

```



```
{% endraw %}
```

INDEX.CSS

```
/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
```

Defining priority order for group collapse or expand

You can use the [priority](#) property to set the priority order for each group which should be collapsed or expanded on resizing. When collapsing, higher priority values are fetched first. When expanding, lower priority values are fetched first.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
  const findOptions = [{ text: "Find", iconCss: "e-icons e-search" }, {
text: "Advanced find", iconCss: "e-icons e-search" }, { text: "Go to",
iconCss: "e-icons e-arrow-right" }];
  const selectOptions = [{ text: "Select All" }, { text: "Select Objects"
}];
  const fontSize = ["8", "9", "10", "11", "12", "14", "16", "18", "20",
"22", "24", "26", "28", "36", "48", "72", "96"];
  const fontStyle = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria
Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe
Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste" priority={2}>
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
```

```

        </RibbonItemDirective>
    </RibbonItemsDirective>
</RibbonCollectionDirective>
<RibbonCollectionDirective>
    <RibbonItemsDirective>
        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
cut", content: "Cut" }}>
            </RibbonItemDirective>
        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
copy", content: "Copy" }}>
            </RibbonItemDirective>
        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
format-painter", content: "Format Painter" }}>
            </RibbonItemDirective>
    </RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
<RibbonGroupDirective header="Font" groupIconCss="e-
icons e-bold" priority={0}>
    <RibbonCollectionsDirective>
        <RibbonCollectionDirective>
            <RibbonItemsDirective>
                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 2 }}>
                    </RibbonItemDirective>
                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize, index: 4 }}>
                    </RibbonItemDirective>
            </RibbonItemsDirective>
        </RibbonCollectionDirective>
        <RibbonCollectionDirective>
            <RibbonItemsDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-bold", content: "Bold" }}>
                    </RibbonItemDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-italic", content: "Italic" }}>
                    </RibbonItemDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-underline", content: "Underline" }}>
                    </RibbonItemDirective>
            </RibbonItemsDirective>
        </RibbonCollectionDirective>
    </RibbonCollectionsDirective>
</RibbonGroupDirective>
<RibbonGroupDirective header="Editor"
groupIconCss="e-icons e-edit" priority={1}>
    <RibbonCollectionsDirective>
        <RibbonCollectionDirective>
            <RibbonItemsDirective>
                <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-search", items:
findOptions, content: "Find" }}>

```

```

        </RibbonItemDirective>
        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-replace", content: "Replace" }}>
        </RibbonItemDirective>
        <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
items: selectOptions, content: "Select" }}>
        </RibbonItemDirective>
        </RibbonItemsDirective>
        </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        </RibbonGroupsDirective>
        </RibbonTabDirective>
        </RibbonTabsDirective>
    </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
    const findOptions = [{ text: "Find", iconCss: "e-icons e-search" }, {
text: "Advanced find", iconCss: "e-icons e-search" }, { text: "Go to",
iconCss: "e-icons e-arrow-right" }];
    const selectOptions = [{ text: "Select All" }, { text: "Select Objects"
}];
    const fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
"20", "22", "24", "26", "28", "36", "48", "72", "96"];
    const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste" priority={2}>
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>

```

```

                                <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
                                splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
format-painter", content: "Format Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Font" groupIconCss="e-
icons e-bold" priority={0}>
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 2 }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize, index: 4 }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-bold", content: "Bold" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-italic", content: "Italic" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-underline", content: "Underline" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Editor"
groupIconCss="e-icons e-edit" priority={1}>
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>

```

```

                                <RibbonItemsDirective>
                                    <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-search", items:
findOptions, content: "Find" }}>
                                </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-replace", content: "Replace" }}>
                                </RibbonItemDirective>
                                    <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
items: selectOptions, content: "Select" }}>
                                </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

Simplified layout

In simplified layout, the Ribbon component organizes the items and groups into a single row by setting the [activeLayout](#) property to [Simplified](#).

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
    const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];

```

```

const tableOptions = [{ text: "Insert Table" }, { text: "This device" },
{ text: "Convert Table" }, { text: "Excel SpreadSheet" }];
return (
  <RibbonComponent id="ribbon" activeLayout='Simplified'>
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home">
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Clipboard">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective
type="SplitButton"
                    splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            <RibbonCollectionDirective>
              <RibbonItemsDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                </RibbonItemDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                </RibbonItemDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
    <RibbonTabDirective header="Insert">
      <RibbonGroupsDirective>
        <RibbonGroupDirective header="Tables">
          <RibbonCollectionsDirective>
            <RibbonCollectionDirective>
              <RibbonItemsDirective>
                <RibbonItemDirective type="DropDown"
dropDownSettings={{ iconCss: "e-icons e-table", items: tableOptions, content:
"Table" }}>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;

```

```
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
    const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text:
"This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
    return (
        <RibbonComponent id="ribbon" activeLayout='Simplified'>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton"
                                        splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
                <RibbonTabDirective header="Insert">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Tables">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        </RibbonComponent>
    );
}
```

```

                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="DropDown"
dropDownSettings={{ iconCss: "e-icons e-table", items: tableOptions, content:
"Table" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                </RibbonComponent>
                            );
                        }
                    export default App;
                    ReactDOM.render(<App />, document.getElementById("element"));
                    {% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

Enabling group overflow popup

You can use the [enableGroupOverflow](#) property to add a separate popup for the overflow items in the group while resizing. The overflow items will appear in the standard common popup, located at the right end of the tab content if it is set to `false`.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize, RibbonColorPicker, Inject } from "@syncfusion/ej2-react-
ribbon";
function App() {
    const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
    const findOptions = [{ text: "Find", iconCss: "e-icons e-search" }, {
text: "Advanced find", iconCss: "e-icons e-search" }, { text: "Go to",
iconCss: "e-icons e-arrow-right" }];
    const selectOptions = [{ text: "Select All" }, { text: "Select Objects"
}];

```



```

const fontSize = ["8", "9", "10", "11", "12", "14", "16", "18", "20",
"22", "24", "26", "28", "36", "48", "72", "96"];
const fontStyle = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria
Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe
Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"];
return (
  <RibbonComponent id="ribbon" activeLayout="Simplified">
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home">
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          <RibbonCollectionDirective>
            <RibbonItemsDirective>
              <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
cut", content: "Cut" }}>
            </RibbonItemDirective>
              <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
copy", content: "Copy" }}>
            </RibbonItemDirective>
              <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
format-painter", content: "Format Painter" }}>
            </RibbonItemDirective>
          </RibbonItemsDirective>
        </RibbonCollectionDirective>
      </RibbonCollectionsDirective>
    </RibbonGroupDirective>
    <RibbonGroupDirective header="Font" groupIconCss="e-
icons e-bold" enableGroupOverflow={true}>
      <RibbonCollectionsDirective>
        <RibbonCollectionDirective>
          <RibbonItemsDirective>
            <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 2 }}>
          </RibbonItemDirective>
            <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize, index: 4 }}>
          </RibbonItemDirective>
        </RibbonItemsDirective>
      </RibbonCollectionDirective>
    </RibbonCollectionDirective>
  </RibbonItemsDirective>
</RibbonItemDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>

```

```

                                <RibbonItemDirective
type="ColorPicker" allowedSizes={RibbonItemSize.Small} colorPickerSettings={{
value: '#123456' }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-bold", content: "Bold" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-italic", content: "Italic" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-underline", content: "Underline" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Editor"
groupIconCss="e-icons e-edit">
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-search", items:
findOptions, content: "Find" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-replace", content: "Replace" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
items: selectOptions, content: "Select" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonColorPicker]} />
                                </RibbonComponent>
                                );
                                }
                                export default App;
                                ReactDOM.render(<App />, document.getElementById("element"));
                                {% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,

```

```

RibbonItemSize, RibbonColorPicker, Inject } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
    const findOptions = [{ text: "Find", iconCss: "e-icons e-search" }, {
text: "Advanced find", iconCss: "e-icons e-search" }, { text: "Go to",
iconCss: "e-icons e-arrow-right" }];
    const selectOptions = [{ text: "Select All" }, { text: "Select Objects"
}];
    const fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
"20", "22", "24", "26", "28", "36", "48", "72", "96"];
    const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];
    return (
        <RibbonComponent id="ribbon" activeLayout="Simplified">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
cut", content: "Cut" }}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
copy", content: "Copy" }}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} buttonSettings={{ iconCss: "e-icons e-
format-painter", content: "Format Painter" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        <RibbonGroupDirective header="Font" groupIconCss="e-
icons e-bold" enableGroupOverflow={true}>
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>

```

```

                                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontStyle, index: 2 }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="ComboBox"
comboBoxSettings={{ dataSource: fontSize, index: 4 }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective
type="ColorPicker" allowedSizes={RibbonItemSize.Small} colorPickerSettings={{
value: '#123456' }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-bold", content: "Bold" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-italic", content: "Italic" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-underline", content: "Underline" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Editor"
groupIconCss="e-icons e-edit">
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-search", items:
findOptions, content: "Find" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-replace", content: "Replace" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
items: selectOptions, content: "Select" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonColorPicker]} />
                                </RibbonComponent>
        );
    }
    export default App;
    ReactDOM.render(<App />, document.getElementById("element"));
    {% enddraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Minimized State

You can hide the Ribbon contents and display only the tab headers by double-clicking on the tab header. In minimized state, the Ribbon component expands to its normal state when click on the tab header.

You can use the [isMinimized](#) property to change the Ribbon component to minimized state. By default, the value is `false`.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
  const tableOptions = [{ text: "Insert Table" }, { text: "This device" },
{ text: "Convert Table" }, { text: "Excel Spreadsheet" }];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" isMinimized='true'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton"
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionDirective>
          <RibbonItemsDirective>
            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
          </RibbonItemDirective>

```

```

        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
        </RibbonItemDirective>
        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
        </RibbonItemDirective>
    </RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
<RibbonTabDirective header="Insert">
    <RibbonGroupsDirective>
        <RibbonGroupDirective header="Tables">
            <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                        <RibbonItemDirective type="DropDown"
dropDownSettings={{ iconCss: "e-icons e-table", items: tableOptions, content:
"Table" }}>
                        </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
        </RibbonGroupDirective>
    </RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
  const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];
  const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text:
"This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  return (
    <RibbonComponent id="ribbon" isMinimized='true'>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
```

```

        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Clipboard">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective
type="SplitButton"
                                splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            <RibbonCollectionDirective>
              <RibbonItemsDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                </RibbonItemDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                </RibbonItemDirective>
                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
    <RibbonTabDirective header="Insert">
      <RibbonGroupsDirective>
        <RibbonGroupDirective header="Tables">
          <RibbonCollectionsDirective>
            <RibbonCollectionDirective>
              <RibbonItemsDirective>
                <RibbonItemDirective type="DropDown"
dropDownSettings={{ iconCss: "e-icons e-table", items: tableOptions, content:
"Table" }}>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

File Menu

The Ribbon component provides a built-in file menu that allows you to add menu items for performing specific actions. The file menu can be enabled by setting the [fileMenu](#) property.

Visibility

You can show the file menu by setting the [visible](#) property to `true`. By default, the file menu is hidden.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
  const fileOptions = [{ text: "New", iconCss: "e-icons e-file-new", id:
"new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  { text: "Save as", iconCss: "e-icons e-save", id: "save" }]
  return (
    <RibbonComponent id="ribbon" fileMenu={{ visible: true, menuItems:
fileOptions}}>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  )
}

```



```

        <Inject services={[RibbonFileMenu]} />
    </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
import { MenuItemModel } from '@syncfusion/ej2-react-navigations';
function App() {
    const fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-
file-new", id: "new" },
    { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
    { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
    { text: "Save as", iconCss: "e-icons e-save", id: "save" } ]
    return (
        <RibbonComponent id="ribbon" fileMenu={{ visible: true, menuItems:
fileOptions}}>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
            <Inject services={[RibbonFileMenu]} />
        </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Adding menu items

The menu items can be added to the file menu using the [menuItems](#) property as an array of [MenuItemModel](#).

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, RibbonColorPicker, Inject } from "@syncfusion/ej2-
react-ribbon";
function App() {
  const fileOptions = [{ text: "New", iconCss: "e-icons e-file-new", id:
"new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  { text: "Save as", iconCss: "e-icons e-save", id: "save" }]
  return (
    <RibbonComponent id="ribbon" fileMenu={{ visible: true, menuItems:
fileOptions}}>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="ColorPicker" colorPickerSettings={{ value: "#123456" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionDirective>
            <RibbonItemsDirective>
              <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
              </RibbonItemDirective>
              <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
            </RibbonItemDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>

```

```

        </RibbonItemsDirective>
        </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        </RibbonGroupsDirective>
        </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonFileMenu, RibbonColorPicker]} />
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, RibbonColorPicker, Inject } from "@syncfusion/ej2-
react-ribbon";
import { MenuItemModel } from '@syncfusion/ej2-react-navigations';
function App() {
  const fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-
file-new", id: "new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  { text: "Save as", iconCss: "e-icons e-save", id: "save" } ]
  return (
    <RibbonComponent id="ribbon" fileMenu={{ visible: true, menuItems:
fileOptions}}>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="ColorPicker" colorPickerSettings={{ value: "#123456" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                    </RibbonItemDirective>

```

```

        </RibbonItemsDirective>
        </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        </RibbonGroupsDirective>
        </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonFileMenu, RibbonColorPicker]} />
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Open submenu on click

You can open the submenu on menu item click, by setting the [showItemOnClick](#) property to `true`. By default, the submenu will open on mouse hover.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
  const fileOptions = [{ text: "New", iconCss: "e-icons e-file-new", id:
"new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  {
    text: "Save as", iconCss: "e-icons e-save", id: "save",
    items: [
      { text: "Microsoft Word (.docx)", iconCss: "sf-icon-word", id:
"newword" },
      { text: "Microsoft Word 97-2003(.doc)", iconCss: "sf-icon-word",
id: "oldword" },
      { text: "Download as PDF", iconCss: "e-icons e-export-pdf", id:
"pdf" }]
    }
  ];
}

```

```

    ]]
    return (
      <RibbonComponent id="ribbon" fileMenu={{ visible:
true,showItemOnClick: true, menuItems: fileOptions}}>
        <RibbonTabsDirective>
          <RibbonTabDirective header="Home">
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                    </RibbonItemDirective>
                      <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
      <Inject services={[RibbonFileMenu]} />
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
import { MenuItemModel } from '@syncfusion/ej2-react-navigations';
function App() {
  const fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-
file-new", id: "new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  {
    text: "Save as", iconCss: "e-icons e-save", id: "save",
    items: [
      { text: "Microsoft Word (.docx)", iconCss: "sf-icon-word", id:
"newword" },
      { text: "Microsoft Word 97-2003(.doc)", iconCss: "sf-icon-word",
id: "oldword" },
    ]
  }
];
}

```

```

        { text: "Download as PDF", iconCss: "e-icons e-export-pdf", id:
"pdf" }}
    ]]
    return (
        <RibbonComponent id="ribbon" fileMenu={{ visible:
true,showItemOnClick: true, menuItems: fileOptions}}>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
            <Inject services={[RibbonFileMenu]} />
        </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

Custom header text

You can define the file menu header text content by using the [text](#) property.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,

```

```

RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
    const fileOptions = [{ text: "New", iconCss: "e-icons e-file-new", id:
    "new" }]
    return (
        <RibbonComponent id="ribbon" fileMenu={{ visible: true,text: 'App',
        menuItems: fileOptions}}>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
            <Inject services={[RibbonFileMenu]} />
        </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective } from
"@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
import { MenuItemModel } from '@syncfusion/ej2-react-navigations';
function App() {
    const fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-
file-new", id: "new" }]
    return (
        <RibbonComponent id="ribbon" fileMenu={{ visible: true,text: 'App',
        menuItems: fileOptions}}>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">

```

```

        <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
                <RibbonCollectionsDirective>
                    <RibbonCollectionDirective>
                        <RibbonItemsDirective>
                            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                        </RibbonItemDirective>
                            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                        </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
        </RibbonGroupDirective>
    </RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonFileMenu]} />
</RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

Ribbon Backstage

The Ribbon supports backstage view which is an addition to the traditional file menu. It displays information like application settings, user details, etc. The backstage view can be enabled by setting the [backStageMenu](#) property.

The backstage view options are displayed on the left, while the content of each option is shown on the right.

Adding backstage items

The menu items can be added to the backstage view by using the [items](#) property. You can show the backstage view by setting the [visible](#) property to `true`. By default, the backstage view is hidden.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";

```



```

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective ,
RibbonBackstage, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
    const backstageSettings = {
        visible: true,
        items: [
            { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() },
            { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
newContentTemplate() },
            { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open',
content: openContentTemplate() }
        ],
        backButton: {
            text: 'Close',
        }
    }
    function homeContentTemplate () {
        return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
    }

    function newContentTemplate () {
        return (
            "<div id='new-content' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div></div> "
        )
    }

    function openContentTemplate () {
        return (
            "<div style='padding: 20px;'><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
12px'> Use the full functionality of Ribbon
</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
style='display: block; font-size: 14px'> Protect Document </span><span
style='font-size: 12px'>To prevent accidental changes, this document has been
set to open as view-only.</span></td></tr></tbody></table></div></div>"
        )
    }
    return (

```

```

    <RibbonComponent id="backstage-ribbon" backStageMenu={
backstageSettings }>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Paragraph">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
                      </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
      <Inject services={[RibbonBackstage]} />
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonBackstage, Inject, BackStageMenuModel } from "@syncfusion/ej2-react-
ribbon";
function App() {
  const backstageSettings: BackStageMenuModel = {
    visible: true,
    items: [
      { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() },
      { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
newContentTemplate() },
      { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open',
content: openContentTemplate() }
    ],
    backButton: {
      text: 'Close',

```

```

    }
  }
  function homeContentTemplate () {
    return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
  }

  function newContentTemplate () {
    return (
      "<div id='new-content' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div></div> "
    )
  }

  function openContentTemplate () {
    return (
      "<div style='padding: 20px;'><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
12px'> Use the full functionality of Ribbon
</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
style='display: block; font-size: 14px'> Protect Document </span><span
style='font-size: 12px'>To prevent accidental changes, this document has been
set to open as view-only.</span></td></tr></tbody></table></div></div>"
    )
  }

  return (
    <RibbonComponent id="backstage-ribbon"
backStageMenu={backstageSettings}>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Paragraph">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
                    </RibbonItemDirective>

```

```

        </RibbonItemsDirective>
        </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        </RibbonGroupsDirective>
        </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonBackstage]} />
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
/* Represents the styles for Ribbon Backstage */
.e-ribbon-backstage-content{
  width: 550px;
  height: 350px;
}
.section-title {
  font-size: 22px;
}
.new-docs {
  display: flex;
  justify-content: space-around;
  flex-wrap: wrap;
}
.category_container {
  width: 150px;
  padding: 15px;
  text-align: center;
  cursor: pointer;
}
.doc_category_image {
  width: 80px;
  height: 100px;
  background-color: #fff;
  border: 1px solid rgb(125, 124, 124);
  text-align: center;
  overflow: hidden;
  margin: 0px auto 10px;
}
.doc_category_text {
  font-size: 16px;
}

```

```

}
.section-content {
  padding: 12px 0px;
  cursor: pointer;
}
.doc_icon {
  font-size: 16px;
  padding: 0px 10px;
}
.category_container:hover, .section-content:hover {
  background-color: #dfdfff;
  border-radius: 5px;
  transition: all 0.3s;
}

```

Adding footer items

You can use the [isFooter](#) property in the `items` collection to add the backstage view footer items. By default, the value is false.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective ,
RibbonBackstage, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
  const backstageSettings = {
    visible: true,
    items: [
      { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() },
      { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
newContentTemplate() },
      { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open',
content: openContentTemplate() },
      { separator: true, isFooter: true },
      { id: 'account', text: 'Account', isFooter: true, content:
accountContent() }
    ],
    backButton: {
      text: 'Close',
    }
  }
  function homeContentTemplate () {
    return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span

```

```

style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
    }

    function newContentTemplate () {
        return (
            "<div id='new-content' style='padding: 20px;'><div id='new-
            section' class='new-wrapper'><div class='section-title'> New </div><div
            class='category_container'><div class='doc_category_image'></div><span
            class='doc_category_text'> New document </span></div></div></div> "
        )
    }
    function openContentTemplate () {
        return (
            "<div style='padding: 20px;'><div class='section-content'
            style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
            class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
            block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
            12px'> Use the full functionality of Ribbon
            </span></td></tr></tbody></table></div><div class='section-content'
            style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
            class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
            style='display: block; font-size: 14px'> Protect Document </span><span
            style='font-size: 12px'>To prevent accidental changes, this document has been
            set to open as view-only.</span></td></tr></tbody></table></div></div>"
        )
    }
    function accountContent () {
        return (
            "<div style='padding: 20px;'><div class='section-content'
            style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
            class='doc_icon e-icons e-people'></span> </td><td><span style='display:
            block; font-size: 14px'>Account type</span><span style='font-size:
            12px'>Administrator</span></td></tr></tbody></table></div><div
            class='section-content' style='padding: 12px 0px; cursor:
            pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-
            password'></span> </td><td><span style='display: block; font-size:
            14px'>Password protected</span><span style='font-size:
            12px'>Yes</span></td></tr></tbody></table></div></div>"
        )
    }
    return (
        <RibbonComponent id="backstage-ribbon" backStageMenu={
        backstageSettings }>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home" >
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Paragraph">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
                                        buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
                                        buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                        </RibbonItemDirective>

```

```

                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonBackstage]} />
                                </RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonBackstage , Inject, BackStageMenuModel } from "@syncfusion/ej2-react-
ribbon";
function App() {
    const backstageSettings: BackStageMenuModel = {
        visible: true,
        items: [
            { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() },
            { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
newContentTemplate() },
            { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open',
content: openContentTemplate() },
            { separator: true, isFooter: true },
            { id: 'account', text: 'Account', isFooter: true, content:
accountContent() }
        ],
        backButton: {
            text: 'Close',
        }
    }
    function homeContentTemplate () {
        return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span

```

```

style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
    }

    function newContentTemplate () {
        return (
            "<div id='new-content' style='padding: 20px;'><div id='new-
            section' class='new-wrapper'><div class='section-title'> New </div><div
            class='category_container'><div class='doc_category_image'></div><span
            class='doc_category_text'> New document </span></div></div></div> "
        )
    }
    function openContentTemplate () {
        return (
            "<div style='padding: 20px;'><div class='section-content'
            style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
            class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
            block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
            12px'> Use the full functionality of Ribbon
            </span></td></tr></tbody></table></div><div class='section-content'
            style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
            class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
            style='display: block; font-size: 14px'> Protect Document </span><span
            style='font-size: 12px'>To prevent accidental changes, this document has been
            set to open as view-only.</span></td></tr></tbody></table></div></div>"
        )
    }
    function accountContent () {
        return (
            "<div style='padding: 20px;'><div class='section-content'
            style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
            class='doc_icon e-icons e-people'></span> </td><td><span style='display:
            block; font-size: 14px'>Account type</span><span style='font-size:
            12px'>Administrator</span></td></tr></tbody></table></div><div
            class='section-content' style='padding: 12px 0px; cursor:
            pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-
            password'></span> </td><td><span style='display: block; font-size:
            14px'>Password protected</span><span style='font-size:
            12px'>Yes</span></td></tr></tbody></table></div></div>"
        )
    }
    return (
        <RibbonComponent id="backstage-ribbon"
        backStageMenu={backstageSettings}>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home" >
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Paragraph">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
                                        buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
                                        buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                        </RibbonItemDirective>

```



```

                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonBackstage]} />
                                </RibbonComponent>
        );
    }
    export default App;
    ReactDOM.render(<App />, document.getElementById("element"));
    {% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
/* Represents the styles for Ribbon Backstage */
.e-ribbon-backstage-content{
    width: 550px;
    height: 350px;
}
.section-title {
    font-size: 22px;
}
.new-docs {
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
}
.category_container {
    width: 150px;
    padding: 15px;
    text-align: center;
    cursor: pointer;
}
.doc_category_image {
    width: 80px;
    height: 100px;
    background-color: #fff;
    border: 1px solid rgb(125, 124, 124);
    text-align: center;
    overflow: hidden;
    margin: 0px auto 10px;
}

```

```

}
.doc_category_text {
  font-size: 16px;
}
.section-content {
  padding: 12px 0px;
  cursor: pointer;
}
.doc_icon {
  font-size: 16px;
  padding: 0px 10px;
}
.category_container:hover, .section-content:hover {
  background-color: #dfdfff;
  border-radius: 5px;
  transition: all 0.3s;
}

```

Adding separator

The separators are horizontal lines used to separate the backstage view items. You can use the [separator](#) property to split the menu items.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective ,
RibbonBackstage, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
  const backstageSettings = {
    visible: true,
    items: [
      { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() },
      { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
newContentTemplate() },
      { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open',
content: openContentTemplate() },
      { separator: true },
      { text: 'Print', content: printContent() }
    ],
    backButton: {
      text: 'Close',
    }
  }
  function homeContentTemplate () {
    return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span

```

```

style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
    }

    function newContentTemplate () {
        return (
            "<div id='new-content' style='padding: 20px;'><div id='new-
            section' class='new-wrapper'><div class='section-title'> New </div><div
            class='category_container'><div class='doc_category_image'></div><span
            class='doc_category_text'> New document </span></div></div></div> "
        )
    }
    function openContentTemplate () {
        return (
            "<div style='padding: 20px;'><div class='section-content'
            style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
            class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
            block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
            12px'> Use the full functionality of Ribbon
            </span></td></tr></tbody></table></div><div class='section-content'
            style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
            class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
            style='display: block; font-size: 14px'> Protect Document </span><span
            style='font-size: 12px'>To prevent accidental changes, this document has been
            set to open as view-only.</span></td></tr></tbody></table></div></div>"
        )
    }
    function printContent () {
        return (
            "<div style='min-width: 300px; padding: 20px;'><h2>Print this
            document</h2><button id='togglebtn' class='e-control e-btn e-lib e-flat e-
            primary'><span class='e-btn-icon e-btn-sb-icons e-icons e-print e-icon-
            left'></span>Print</button></div>"
        )
    }
    return (
        <RibbonComponent id="backstage-ribbon" backStageMenu={
backstageSettings }>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home" >
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Paragraph">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonGroupsDirective>
                        </RibbonTabDirective>
                    </RibbonTabsDirective>
                </RibbonComponent>
            )
    }

```

```

        </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        </RibbonGroupsDirective>
        </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonBackstage]} />
    </RibbonComponent>

    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonBackstage, Inject, BackStageMenuModel } from "@syncfusion/ej2-react-ribbon";
function App() {
    const backstageSettings: BackStageMenuModel = {
        visible: true,
        items: [
            { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() },
            { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
newContentTemplate() },
            { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open',
content: openContentTemplate() },
            { separator: true },
            { text: 'Print', content: printContent() }
        ],
        backButton: {
            text: 'Close',
        }
    }
    function homeContentTemplate () {
        return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
    }

    function newContentTemplate () {
        return (

```

```

        "<div id='new-content' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div></div> "
    )
}
function openContentTemplate () {
    return (
        "<div style='padding: 20px;'><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
12px'> Use the full functionality of Ribbon
</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
style='display: block; font-size: 14px'> Protect Document </span><span
style='font-size: 12px'>To prevent accidental changes, this document has been
set to open as view-only.</span></td></tr></tbody></table></div></div>"
    )
}
function printContent () {
    return (
        "<div style='min-width: 300px; padding: 20px;'><h2>Print this
document</h2><button id='togglebtn' class='e-control e-btn e-lib e-flat e-
primary'><span class='e-btn-icon e-btn-sb-icons e-icons e-print e-icon-
left'></span>Print</button></div>"
    )
}
return (
    <RibbonComponent id="backstage-ribbon"
backStageMenu={backstageSettings}>
        <RibbonTabsDirective>
            <RibbonTabDirective header="Home" >
                <RibbonGroupsDirective>
                    <RibbonGroupDirective header="Paragraph">
                        <RibbonCollectionsDirective>
                            <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
                                </RibbonItemDirective>
                            </RibbonItemsDirective>
                        </RibbonCollectionDirective>
                    </RibbonCollectionsDirective>
                </RibbonGroupDirective>
            </RibbonGroupsDirective>
        </RibbonTabDirective>
    </RibbonTabsDirective>
    <Inject services={[RibbonBackstage]} />
</RibbonComponent>

```

```

    );
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById("element"));
  {% enddraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
/* Represents the styles for Ribbon Backstage */
.e-ribbon-backstage-content{
  width: 350px;
  height: 530px;
}
.section-title {
  font-size: 22px;
}
.new-docs {
  display: flex;
  justify-content: space-around;
  flex-wrap: wrap;
}
.category_container {
  width: 150px;
  padding: 15px;
  text-align: center;
  cursor: pointer;
}
.doc_category_image {
  width: 80px;
  height: 100px;
  background-color: #fff;
  border: 1px solid rgb(125, 124, 124);
  text-align: center;
  overflow: hidden;
  margin: 0px auto 10px;
}
.doc_category_text {
  font-size: 16px;
}
.section-content {
  padding: 12px 0px;
  cursor: pointer;
}
.doc_icon {
  font-size: 16px;
  padding: 0px 10px;
}

```

```
.category_container:hover, .section-content:hover {
  background-color: #dfdfdf;
  border-radius: 5px;
  transition: all 0.3s;
}
```

Back button

You can use the [backButton](#) property to customize the text and icon of the close button using the [text](#) and [iconCss](#) property. You can show the back button by setting the [visible](#) property to `true`.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective ,
RibbonBackstage, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
  const backstageSettings = {
    visible: true,
    items: [
      { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() }
    ],
    backButton: {
      text: 'Close',
    }
  }
  function homeContentTemplate () {
    return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
  }

  return (
    <RibbonComponent id="backstage-ribbon" backStageMenu={
backstageSettings }>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Paragraph">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                    </RibbonItemDirective>
```

```

                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonBackstage]} />
                                </RibbonComponent>
        );
    }
    export default App;
    ReactDOM.render(<App />, document.getElementById("element"));
    {% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonBackstage, Inject, BackStageMenuModel } from "@syncfusion/ej2-react-
ribbon";
function App() {
    const backstageSettings: BackStageMenuModel = {
        visible: true,
        items: [
            { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() }
        ],
        backButton: {
            text: 'Close',
        }
    }
    function homeContentTemplate () {
        return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
    }
    return (

```



```

    <RibbonComponent id="backstage-ribbon"
backStageMenu={backstageSettings}>
    <RibbonTabsDirective>
    <RibbonTabDirective header="Home" >
    <RibbonGroupsDirective>
    <RibbonGroupDirective header="Paragraph">
    <RibbonCollectionsDirective>
    <RibbonCollectionDirective>
    <RibbonItemsDirective>
    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
    </RibbonItemDirective>
    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
    </RibbonItemDirective>
    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
    </RibbonItemDirective>
    </RibbonItemsDirective>
    </RibbonCollectionDirective>
    </RibbonCollectionsDirective>
    </RibbonGroupDirective>
    </RibbonGroupsDirective>
    </RibbonTabDirective>
    </RibbonTabsDirective>
    <Inject services={[RibbonBackstage]} />
    </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
/* Represents the styles for Ribbon Backstage */
.e-ribbon-backstage-content{
  width: 550px;
  height: 350px;
}
.section-title {
  font-size: 22px;
}
.new-docs {
  display: flex;
  justify-content: space-around;
  flex-wrap: wrap;
}

```

```

}
.category_container {
  width: 150px;
  padding: 15px;
  text-align: center;
  cursor: pointer;
}
.doc_category_image {
  width: 80px;
  height: 100px;
  background-color: #fff;
  border: 1px solid rgb(125, 124, 124);
  text-align: center;
  overflow: hidden;
  margin: 0px auto 10px;
}
.doc_category_text {
  font-size: 16px;
}
}
.section-content {
  padding: 12px 0px;
  cursor: pointer;
}
}
.doc_icon {
  font-size: 16px;
  padding: 0px 10px;
}
}
.category_container:hover, .section-content:hover {
  background-color: #dfdfff;
  border-radius: 5px;
  transition: all 0.3s;
}
}

```

Backstage target

The [target](#) property specifies the element selector in which backstage will be displayed. The target element should have the position as relative, else the backstage will be positioned nearest to the relative element. By default, the backstage is positioned to ribbon element.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective ,
RibbonBackstage, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
  const backstageSettings = {
    target: '#targetElement',
    visible: true,
    items: [
      { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() },
      { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
newContentTemplate() },

```

```

        { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open',
content: openContentTemplate() }
    ],
    backButton: {
        text: 'Close',
    }
}
function homeContentTemplate () {
    return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
}

function newContentTemplate () {
    return (
        "<div id='new-content' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div></div> "
    )
}
function openContentTemplate () {
    return (
        "<div style='padding: 20px;'><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
12px'> Use the full functionality of Ribbon
</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
style='display: block; font-size: 14px'> Protect Document </span><span
style='font-size: 12px'>To prevent accidental changes, this document has been
set to open as view-only.</span></td></tr></tbody></table></div></div>"
    )
}
return (
    <div>
        <RibbonComponent id="backstage-ribbon" backStageMenu={
backstageSettings }>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home" >
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Paragraph">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                            </RibbonItemDirective>

```

```

                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                </RibbonItemDirective>
                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste"
}}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonBackstage]} />
                                </RibbonComponent>
                                <div id="targetElement"></div>
                                </div>
        );
    }
    export default App;
    ReactDOM.render(<App />, document.getElementById("element"));
    {% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonBackstage, Inject, BackStageMenuModel } from "@syncfusion/ej2-react-
ribbon";
function App() {
    const backstageSettings: BackStageMenuModel = {
        target: '#targetElement',
        visible: true,
        items: [
            { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() },
            { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
newContentTemplate() },
            { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open',
content: openContentTemplate() }
        ],
        backButton: {
            text: 'Close',
        }
    }
    function homeContentTemplate () {
        return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-

```

```

wrapper')<div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
    }

    function newContentTemplate () {
        return (
            "<div id='new-content' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div></div> "
        )
    }
    function openContentTemplate () {
        return (
            "<div style='padding: 20px;'><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
12px'> Use the full functionality of Ribbon
</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
style='display: block; font-size: 14px'> Protect Document </span><span
style='font-size: 12px'>To prevent accidental changes, this document has been
set to open as view-only.</span></td></tr></tbody></table></div></div>"
        )
    }
    return (
        <div>
            <RibbonComponent id="backstage-ribbon"
backStageMenu={backstageSettings}>
                <RibbonTabsDirective>
                    <RibbonTabDirective header="Home" >
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Paragraph">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste"
}}>
                                                </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>

```

```

        </RibbonGroupsDirective>
        </RibbonTabDirective>
    </RibbonTabsDirective>
    <Inject services={[RibbonBackstage]} />
    </RibbonComponent>
    <div id="targetElement"></div>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
/* Represents the styles for Ribbon Backstage */
.e-ribbon-backstage-content{
    width: 550px;
    height: 350px;
}
.section-title {
    font-size: 22px;
}
.new-docs {
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
}
.category_container {
    width: 150px;
    padding: 15px;
    text-align: center;
    cursor: pointer;
}
.doc_category_image {
    width: 80px;
    height: 100px;
    background-color: #fff;
    border: 1px solid rgb(125, 124, 124);
    text-align: center;
    overflow: hidden;
    margin: 0px auto 10px;
}
.doc_category_text {
    font-size: 16px;
}
.section-content {

```

```
padding: 12px 0px;
cursor: pointer;
}
.doc_icon {
font-size: 16px;
padding: 0px 10px;
}
.category_container:hover, .section-content:hover {
background-color: #dfdfff;
border-radius: 5px;
transition: all 0.3s;
}
```

Template

You can use the [template](#) property to modify the backstage view menu items and their contents.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective ,
RibbonBackstage, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
let ribbonEle = null;
React.useEffect(() => {
ribbonEle = document.getElementById('ribbon');
const backstageButton = document.querySelector('.e-ribbon-backstage')
;

if (backstageButton) {
backstageButton.addEventListener('click', displayPopup);
}
}, [])
const backstageSettings = {
visible: true,
template: homeContentTemplate()
}
function displayPopup() {
let backstagePopup =
ribbonEle.querySelector('#ribbon_backstagepopup');
if (backstagePopup) {
backstagePopup.style.display = 'block';
}
}
function contentClick (id) {
let content = ribbonEle.querySelector('.content-open');
if(content) { content.classList.replace('content-open', 'content-
close'); }
ribbonEle.querySelector('#' + id + '-wrapper').classList.add('content-
open');
}
function closeContent () {
ribbonEle.querySelector('#ribbon_backstagepopup').style.display =
'none'
}
```

```

    }
    function homeContentTemplate () {
        return "<div id='temp-content' style='width: 550px; height: 350px;
display: flex'><div id='items-wrapper' style='width: 130px; height:100%;
background: #779de8;'><ul><li id='close' onclick='closeContent()'><span
class='e-icons e-close'></span>Close</li><li id='new'
onclick='contentClick('new')'><span class='e-icons e-file-
new'></span>New</li><li id='open' onclick='contentClick('open')'><span
class='e-icons e-folder-open'></span>Open</li><li id='save'
onclick='contentClick('save')'><span class='e-icons e-
save'></span>Save</li></ul></div><div id='content-wrapper'><div id='new-
wrapper' class='content-open' style='padding: 20px;'><div id='new-section'
class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div
id='save-wrapper' class='content-close' style='padding: 20px;'><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-save'></span>
</td><td><span style='display: block; font-size: 14px'> Save as </span><span
style='font-size: 12px'> Save as copy online
</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-rename'></span> </td><td><span style='display:
block; font-size: 14px'> Rename </span><span style='font-size: 12px'>Rename
this file. </span></td></tr></tbody></table></div></div><div id='open-
wrapper' class='content-close' style='padding: 20px;'><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Classic_layout.docx </span><span style='font-size:
12px'> EJ2 >> Components >> Navigations >> Ribbon >> layouts
</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Simplified_layout.docx </span><span style='font-
size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >> layouts
</span></td></tr></tbody></table></div></div></div></div>"
    }

    return (
        <RibbonComponent id="ribbon" backstageMenu={backstageSettings}>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home" >
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Paragraph">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>

```



```

        </RibbonItemDirective>
        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
            </RibbonItemDirective>
        </RibbonItemsDirective>
    </RibbonCollectionDirective>
</RibbonCollectionsDirective>
    </RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
    <Inject services={[RibbonBackstage]} />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonBackstage, Inject, BackStageMenuModel } from "@syncfusion/ej2-react-
ribbon";
function App() {
    let ribbonEle: HTMLElement | null = null;
    React.useEffect(()=>{
        ribbonEle = document.getElementById('ribbon') as HTMLElement;
        const backstageButton = document.querySelector('.e-ribbon-backstage')
as HTMLElement;
        if (backstageButton) {
            backstageButton.addEventListener('click', displayPopup);
        }
    }, [])
    const backstageSettings: BackStageMenuModel = {
        visible: true,
        template: homeContentTemplate()
    }
    function displayPopup() {
        let backstagePopup =
ribbonEle.querySelector('#ribbon_backstagepopup') as HTMLElement;
        if (backstagePopup) {
            backstagePopup.style.display = 'block';
        }
    }
    function contentClick (id: string) {
        let content = ribbonEle.querySelector('.content-open') as
HTMLElement;
        if (content) { content.classList.replace('content-open', 'content-
close'); }
    }
}

```

```

        (ribbonEle.querySelector('#' + id + '-wrapper') as
HTMLInputElement).classList.add('content-open');
    }
    function closeContent () {
        (ribbonEle.querySelector('#ribbon_backstagepopup') as
HTMLInputElement).style.display = 'none'
    }
    function homeContentTemplate () {
        return "<div id='temp-content' style='width: 550px; height: 350px;
display: flex'><div id='items-wrapper' style='width: 130px; height:100%;
background: #779de8;'><ul><li id='close' onclick='closeContent()'><span
class='e-icons e-close'></span>Close</li><li id='new'
onclick='contentClick('new')'><span class='e-icons e-file-
new'></span>New</li><li id='open' onclick='contentClick('open')'><span
class='e-icons e-folder-open'></span>Open</li><li id='save'
onclick='contentClick('save')'><span class='e-icons e-
save'></span>Save</li></ul></div><div id='content-wrapper'><div id='new-
wrapper' class='content-open' style='padding: 20px;'><div id='new-section'
class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div
id='save-wrapper' class='content-close' style='padding: 20px;'><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-save'></span>
</td><td><span style='display: block; font-size: 14px'> Save as </span><span
style='font-size: 12px'> Save as copy online
</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-rename'></span> </td><td><span style='display:
block; font-size: 14px'> Rename </span><span style='font-size: 12px'>Rename
this file. </span></td></tr></tbody></table></div><div id='open-
wrapper' class='content-close' style='padding: 20px;'><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Classic_layout.docx </span><span style='font-size:
12px'> EJ2 >> Components >> Navigations >> Ribbon >> layouts
</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Simplified_layout.docx </span><span style='font-
size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >> layouts
</span></td></tr></tbody></table></div></div></div></div>"
    }

    return (
        <RibbonComponent id="ribbon" backStageMenu={backstageSettings}>
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home" >
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Paragraph">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>

```

```

                                <RibbonItemsDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                    </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                    </RibbonItemDirective>
                                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                </RibbonGroupsDirective>
            </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonBackstage]} />
    </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
/* Represents the styles for Ribbon Backstage */
.e-ribbon-backstage-content{
    width: 550px;
    height: 350px;
}
.section-title {
    font-size: 22px;
}
.new-docs {
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
}
.category_container {
    width: 150px;
    padding: 15px;
    text-align: center;
    cursor: pointer;
}
.doc_category_image {

```

```
width: 80px;
height: 100px;
background-color: #fff;
border: 1px solid rgb(125, 124, 124);
text-align: center;
overflow: hidden;
margin: 0px auto 10px;
}
.doc_category_text {
  font-size: 16px;
}
.section-content {
  padding: 12px 0px;
  cursor: pointer;
}
.doc_icon {
  font-size: 16px;
  padding: 0px 10px;
}
.category_container:hover, .section-content:hover {
  background-color: #dfdfdf;
  border-radius: 5px;
  transition: all 0.3s;
}
#targetElement{
  width: 500px;
  height: 500px;
}
#items-wrapper ul {
  padding: 0;
  margin: 0;
}
#items-wrapper li {
  height: 38px;
  font-size: 16px;
  list-style: none;
  cursor: pointer;
  text-align: center;
  padding-top: 10px;
}
#items-wrapper li span {
  margin-right: 15px;
  font-size: 14px;
}
#items-wrapper ul li:hover{
  background-color: #a5bff4;
}
#content-wrapper .content-close{
  display: none;
}
#content-wrapper .content-open{
  display: block;
}
```

Setting width and height

You can customize the height and width of the backstage view using the [height](#) and [width](#) property. By default, dimensions are set based on the content added.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective ,
RibbonBackstage, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
  const backstageSettings = {
    visible: true,
    items: [
      { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() },
      { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
newContentTemplate() },
      { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open',
content: openContentTemplate() }
    ],
    backButton: {
      text: 'Close',
    }
  }
  function homeContentTemplate () {
    return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
  }

  function newContentTemplate () {
    return (
      "<div id='new-content' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div></div> "
    )
  }

  function openContentTemplate () {
    return (
      "<div style='padding: 20px;'><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
12px'> Use the full functionality of Ribbon
</span></td></tr></tbody></table></div><div class='section-content'

```

```

style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
style='display: block; font-size: 14px'> Protect Document </span><span
style='font-size: 12px'>To prevent accidental changes, this document has been
set to open as view-only.</span></td></tr></tbody></table></div></div>"
    )
  }
  return (
    <RibbonComponent id="backstage-ribbon" backStageMenu={
backstageSettings }>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Paragraph">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
      <Inject services={[RibbonBackstage]} />
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonBackstage, Inject, BackStageMenuModel } from "@syncfusion/ej2-react-
ribbon";
function App() {
  const backstageSettings: BackStageMenuModel = {
    visible: true,
    items: [

```

```

        { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
homeContentTemplate() },
        { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
newContentTemplate() },
        { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open',
content: openContentTemplate() }
    ],
    backButton: {
        text: 'Close',
    }
}

function homeContentTemplate () {
    return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-
content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
}

function newContentTemplate () {
    return (
        "<div id='new-content' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div></div> "
    )
}

function openContentTemplate () {
    return (
        "<div style='padding: 20px;'><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
12px'> Use the full functionality of Ribbon
</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
style='display: block; font-size: 14px'> Protect Document </span><span
style='font-size: 12px'>To prevent accidental changes, this document has been
set to open as view-only.</span></td></tr></tbody></table></div></div>"
    )
}

return (
    <RibbonComponent id="backstage-ribbon"
backStageMenu={backstageSettings}>
        <RibbonTabsDirective>
            <RibbonTabDirective header="Home" >
                <RibbonGroupsDirective>
                    <RibbonGroupDirective header="Paragraph">
                        <RibbonCollectionsDirective>
                            <RibbonCollectionDirective>
                                <RibbonItemsDirective>

```

```

                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonBackstage]} />
                                </RibbonComponent>
        );
    }
    export default App;
    ReactDOM.render(<App />, document.getElementById("element"));
    {% enddraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
/* Represents the styles for Ribbon Backstage */
.section-title {
    font-size: 22px;
}
.new-docs {
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
}
.category_container {
    width: 150px;
    padding: 15px;
    text-align: center;
    cursor: pointer;
}
.doc_category_image {
    width: 80px;
    height: 100px;
    background-color: #fff;
    border: 1px solid rgb(125, 124, 124);
    text-align: center;
}

```



```

overflow: hidden;
margin: 0px auto 10px;
}
.doc_category_text {
font-size: 16px;
}
.section-content {
padding: 12px 0px;
cursor: pointer;
}
.doc_icon {
font-size: 16px;
padding: 0px 10px;
}
.category_container:hover, .section-content:hover {
background-color: #dfdfdf;
border-radius: 5px;
transition: all 0.3s;
}

```

[Adding Backstage events](#)

Ribbon contextual tabs

The Ribbon Contextual tabs are similar to the Ribbon tabs that are displayed on demand based on their needs, such as an image or a table tabs. It supports adding all built-in and custom ribbon items to perform specific actions.

Visible tabs

You can utilize the [visible](#) property to control the visibility of each contextual tab.

Adding contextual tabs

You can utilize the `RibbonContextualTabsDirective` to add contextual tabs in the Ribbon where you can add multiple tabs based on your needs.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize, RibbonContextualTab, RibbonContextualTabsDirective,
RibbonContextualTabDirective } from "@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id='ribbon'>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
id="clipboardGroup" groupIconCss="e-icons e-paste" showLauncherIcon={true}>
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>

```

```

        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
        </RibbonItemDirective>
        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
        </RibbonItemDirective>
        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
        </RibbonItemDirective>
        </RibbonItemsDirective>
        </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        </RibbonGroupsDirective>
        </RibbonTabDirective>
    </RibbonTabsDirective>
    <RibbonContextualTabsDirective>
        <RibbonContextualTabDirective visible={true} >
            <RibbonTabsDirective>
                <RibbonTabDirective header='Shape Format'
id="ShapeFormat">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Text
decoration" showLauncherIcon={true}>
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-text-header", content:
"Text Header" }}>
                                            </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-text-wrap", content:
"Text Wrap" }}>
                                                </RibbonItemDirective>
                                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-text-annotation",
content: "Text Annotation" }}>
                                                    </RibbonItemDirective>
                                                    </RibbonItemsDirective>
                                                </RibbonCollectionDirective>
                                            </RibbonCollectionsDirective>
                                        </RibbonGroupDirective>
                                    <RibbonGroupDirective header="Accessibility">
                                        <RibbonCollectionsDirective>
                                            <RibbonCollectionDirective>
                                                <RibbonItemsDirective>
                                                    <RibbonItemDirective
type="Button" allowedSizes={RibbonItemSize.Large} buttonSettings={{ iconCss:
"e-icons e-text-alternative", content: "Alt Text" }}>
                                                        </RibbonItemDirective>
                                                        </RibbonItemsDirective>
                                                    </RibbonCollectionDirective>
                                                </RibbonCollectionsDirective>
                                            </RibbonGroupDirective>
                                        </RibbonCollectionsDirective>
                                    </RibbonGroupDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
            </RibbonContextualTabDirective>
        </RibbonContextualTabsDirective>
    </RibbonContextualTabsDirective>
</RibbonContextualTabsDirective>

```

```

                                <RibbonGroupDirective header="Arrange"
showLauncherIcon={true}>
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-bring-forward", content:
"Bring Forward" }}>
                                            </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-send-backward", content:
"Send Backward" }}>
                                            </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-show-hide-panel",
content: "Selection Pane" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
            </RibbonContextualTabDirective>
        </RibbonContextualTabsDirective>
        <Inject services={[RibbonContextualTab]} />
    </RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize, RibbonContextualTab, RibbonContextualTabsDirective,
RibbonContextualTabDirective } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    return (
        <RibbonComponent id='ribbon'>
            <RibbonTabsDirective>
                <RibbonTabDirective header='Home'>
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
id="clipboardGroup" groupIconCss="e-icons e-paste" showLauncherIcon={true}>
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>

```

```

        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
        </RibbonItemDirective>
        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
        </RibbonItemDirective>
        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
        </RibbonItemDirective>
        </RibbonItemsDirective>
    </RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<RibbonContextualTabsDirective>
    <RibbonContextualTabDirective visible={true} >
        <RibbonTabsDirective>
            <RibbonTabDirective header='Shape Format'
id="ShapeFormat">
                <RibbonGroupsDirective>
                    <RibbonGroupDirective header="Text
decoration" showLauncherIcon={true}>
                        <RibbonCollectionsDirective>
                            <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-text-header", content:
"Text Header" }}>
                                    </RibbonItemDirective>
                                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-text-wrap", content:
"Text Wrap" }}>
                                    </RibbonItemDirective>
                                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-text-annotation",
content: "Text Annotation" }}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                    <RibbonGroupDirective header="Accessibility">
                        <RibbonCollectionsDirective>
                            <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                    <RibbonItemDirective
type="Button" allowedSizes={RibbonItemSize.Large} buttonSettings={{ iconCss:
"e-icons e-text-alternative", content: "Alt Text" }}>
                                    </RibbonItemDirective>
                                </RibbonItemsDirective>
                            </RibbonCollectionDirective>
                        </RibbonCollectionsDirective>
                    </RibbonGroupDirective>
                </RibbonGroupDirective>
            </RibbonTabDirective>
        </RibbonTabsDirective>
    </RibbonContextualTabDirective>
</RibbonContextualTabsDirective>

```

```

showLauncherIcon={true}>
    <RibbonGroupDirective header="Arrange"
        <RibbonCollectionsDirective>
            <RibbonCollectionDirective>
                <RibbonItemsDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-bring-forward", content:
"Bring Forward" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-send-backward", content:
"Send Backward" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-show-hide-panel",
content: "Selection Pane" }}>
                    </RibbonItemDirective>
                </RibbonItemsDirective>
            </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
    </RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonContextualTabDirective>
</RibbonContextualTabsDirective>
<Inject services={[RibbonContextualTab]} />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

Selected tabs

By using the [isSelected](#) property you can control the selected state of each contextual tab and indicates which tab is currently active.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";

```

```

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonContextualTab, RibbonContextualTabsDirective,
RibbonContextualTabDirective } from "@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id='ribbon'>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
id="clipboardGroup" groupIconCss="e-icons e-paste" showLauncherIcon={true}>
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
      <RibbonContextualTabsDirective>
        <RibbonContextualTabDirective visible={true}
isSelected={true}>
          <RibbonTabsDirective>
            <RibbonTabDirective header='Styles'>
              <RibbonGroupsDirective>
                <RibbonGroupDirective header="Style"
showLauncherIcon={true}>
                  <RibbonCollectionsDirective>
                    <RibbonCollectionDirective>
                      <RibbonItemsDirective>
                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-style", content: "Style"
}}>
                        </RibbonItemDirective>
                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-font-name", content:
"Text Box" }}>
                        </RibbonItemDirective>
                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-paint-bucket", content:
"Paint" }}>
                        </RibbonItemDirective>
                      </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                  </RibbonCollectionsDirective>
                </RibbonGroupDirective>
              </RibbonGroupsDirective>
            </RibbonTabDirective>
          </RibbonTabsDirective>
        </RibbonContextualTabDirective>
      </RibbonContextualTabsDirective>
    </RibbonComponent>
  );
}

```

```

        </RibbonCollectionDirective>
      </RibbonCollectionsDirective>
    </RibbonGroupDirective>
  </RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonContextualTabDirective>
</RibbonContextualTabsDirective>
<Inject services={[RibbonContextualTab]} />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonContextualTab, RibbonContextualTabsDirective,
RibbonContextualTabDirective } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
  return (
    <RibbonComponent id='ribbon'>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
id="clipboardGroup" groupIconCss="e-icons e-paste" showLauncherIcon={true}>
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
      <RibbonContextualTabsDirective>

```

```

        <RibbonContextualTabDirective visible={true}
isSelected={true}>
            <RibbonTabsDirective>
                <RibbonTabDirective header='Styles'>
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Style"
showLauncherIcon={true}>
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-style", content: "Style"
}}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-font-name", content:
"Text Box" }}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-paint-bucket", content:
"Paint" }}>
                                            </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        </RibbonContextualTabDirective>
    </RibbonContextualTabsDirective>
    <Inject services={[RibbonContextualTab]} />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

Methods

Show tab

You can use the [showTab](#) method to make the specific Contextual tab visible in the Ribbon.

Hide tab

You can use the [hideTab](#) method to hide specific Contextual tab in the Ribbon.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonContextualTab, RibbonContextualTabsDirective,
RibbonContextualTabDirective } from "@syncfusion/ej2-react-ribbon";
import { useRef } from 'react';
function App() {
  let ribbonObj = useRef(null);
  const showContextualTab = () => {
    ribbonObj.current?.showTab('ArrangeView', true);
  }
  const hideContextualTab = () => {
    ribbonObj.current?.hideTab('ArrangeView', true);
  }
  return (
    <div>
      <button className="e-btn" id="show-contextual"
onClick={showContextualTab}> Show tab </button>
      <button className="e-btn" id="hide-contextual"
onClick={hideContextualTab}> Hide tab </button>
      <RibbonComponent id='ribbon' ref={ribbonObj}>
        <RibbonTabsDirective>
          <RibbonTabDirective header='Home'>
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard"
id="clipboardGroup" groupIconCss="e-icons e-paste" showLauncherIcon={true}>
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                    </RibbonItemDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                    </RibbonItemDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                    </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
        <RibbonContextualTabsDirective>
          <RibbonContextualTabDirective>
```

```

        <RibbonTabsDirective>
          <RibbonTabDirective id="ArrangeView"
header="Arrange & View">
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Arrange"
showLauncherIcon={true}>
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-bring-forward", content:
"Bring Forward" }}>
                        </RibbonItemDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-send-backward", content:
"Send Backward" }}>
                        </RibbonItemDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-show-hide-panel",
content: "Selection Pane" }}>
                        </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
      </RibbonContextualTabDirective>
    </RibbonContextualTabsDirective>
    <Inject services={[RibbonContextualTab]} />
  </RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonContextualTab, RibbonContextualTabsDirective,
RibbonContextualTabDirective } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
import { useRef } from 'react';
function App() {
  let ribbonObj = useRef<RibbonComponent>(null);
  const showContextualTab = () => {
    ribbonObj.current?.showTab('ArrangeView', true);
  }
}

```

```

const hideContextualTab = () => {
  ribbonObj.current?.hideTab('ArrangeView', true);
}
return (
  <div>
    <button className="e-btn" id="show-contextual"
onClick={showContextualTab}> Show tab </button>
    <button className="e-btn" id="hide-contextual"
onClick={hideContextualTab}> Hide tab </button>
    <RibbonComponent id='ribbon' ref={ribbonObj}>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
id="clipboardGroup" groupIconCss="e-icons e-paste" showLauncherIcon={true}>
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                    </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                    </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
      <RibbonContextualTabsDirective>
        <RibbonContextualTabDirective>
          <RibbonTabsDirective>
            <RibbonTabDirective id="ArrangeView"
header="Arrange & View">
              <RibbonGroupsDirective>
                <RibbonGroupDirective header="Arrange"
showLauncherIcon={true}>
                  <RibbonCollectionsDirective>
                    <RibbonCollectionDirective>
                      <RibbonItemsDirective>
                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-bring-forward", content:
"Bring Forward" }}>
                        </RibbonItemDirective>
                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-send-backward", content:
"Send Backward" }}>
                        </RibbonItemDirective>
                      </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                  </RibbonCollectionsDirective>
                </RibbonGroupDirective>
              </RibbonGroupsDirective>
            </RibbonTabDirective>
          </RibbonTabsDirective>
        </RibbonContextualTabDirective>
      </RibbonContextualTabsDirective>
    </RibbonComponent>
  </div>
)

```

```

                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-show-hide-panel",
content: "Selection Pane" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                </RibbonContextualTabDirective>
                                </RibbonContextualTabsDirective>
                                <Inject services={[RibbonContextualTab]} />
                                </RibbonComponent>
                            </div>
                        );
                    }
    export default App;
    ReactDOM.render(<App />, document.getElementById("element"));
    {% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
#ribbon {
  margin-top: 20px;
}

```

Ribbon Keytips in React Ribbon component

The Ribbon supports keyboard navigations to interact the ribbon items using the keytips which can be enabled by setting the [enableKeyTips](#) property.

The keytips will be shown when the **Alt + Windows/Command** keys are pressed.

Ribbon items keytip

You can add keytips to all the ribbon items by using the [keyTip](#) property.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel, RibbonItemSize,

```

```

RibbonGroupButtonSelection, RibbonColorPicker, RibbonKeyTip } from
'@syncfusion/ej2-react-ribbon';
function App() {
  let ribbonObj = React.useRef(null);
  const ribbonCreated = () => {
    ribbonObj.current?.ribbonKeyTipModule.showKeyTips('H');
  }
  const gallerySettings = (
    {
      itemCount: 3,
      groups: [{
        itemWidth: '100',
        header: 'Styles',
        items: [{
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {
          content: 'Heading 2'
        }, {
          content: 'Heading 3'
        }, {
          content: 'Heading 4'
        }, {
          content: 'Heading 5'
        }]
      }]
    }
  );

  const tableOptions = [{ text: "Insert Table" }, { text: "Draw Table" }, {
text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
  const fontSize = ["8", "9", "10", "11", "12", "14", "16", "18", "20", "22",
"24", "26", "28", "36", "48", "72", "96"];
  const fontStyle = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria
Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe
Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"];

  return (
    <div>
      <RibbonComponent id='ribbon' ref={ribbonObj} enableKeyTips={true}
created={ribbonCreated}>
        <RibbonTabsDirective>
          <RibbonTabDirective header='Home' keyTip='H'>
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard"
id="clipboard" keyTip='CD' groupIconCss="e-icons e-paste"
showLauncherIcon={true}>
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>

```

```

                                <RibbonItemDirective
type="SplitButton" keyTip='PA' id="pastebtn"
allowedSizes={RibbonItemSize.Large}
                                splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective type="Button"
keyTip='CU' buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
keyTip='CO' buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
keyTip='CS' buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Font" keyTip='FB'
overflowHeader="More Font Options" groupIconCss="e-icons e-bold"
isCollapsible={false} enableGroupOverflow={true} orientation="Row"
cssClass='font-group'>
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective type="ComboBox"
keyTip='O1' comboBoxSettings={{ dataSource: fontStyle, index: 3, label: 'Font
Style', width: '115px', popupWidth: '150px', allowFiltering: true }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="ComboBox"
keyTip='O2' comboBoxSettings={{ dataSource: fontSize, index: 3, label: 'Font
Size', width: '65px', popupWidth: '85px', allowFiltering: true }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective
type="GroupButton" keyTip='GB' allowedSizes={RibbonItemSize.Small}
groupButtonSettings={{selection: RibbonGroupButtonSelection.Single, items:
[{{iconCss: 'e-icons e-bold', keyTip: '1', content: 'Bold', selected: true},
{{iconCss: 'e-icons e-italic', keyTip: '2', content: 'Italic'}, {{iconCss: 'e-
icons e-underline', keyTip: '3', content: 'Underline'}, {{iconCss: 'e-icons e-
strikethrough', keyTip: '4', content: 'Strikethrough'},{{iconCss: 'e-icons e-
change-case', keyTip: '5', content: 'Change Case'}}]}}>
                                </RibbonItemDirective>
                                <RibbonItemDirective
type="ColorPicker" keyTip='CP' allowedSizes={RibbonItemSize.Small}
colorPickerSettings={{value: '#123456'}}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>

```

```

        </RibbonCollectionDirective>
      </RibbonCollectionsDirective>
    </RibbonGroupDirective>
    <RibbonGroupDirective header="Gallery">
      <RibbonCollectionsDirective>
        <RibbonCollectionDirective>
          <RibbonItemsDirective>
            <RibbonItemDirective type="Gallery"
keyTip='GY' gallerySettings={gallerySettings} >
              </RibbonItemDirective>
            </RibbonItemsDirective>
          </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
      </RibbonGroupDirective>
      <RibbonGroupDirective header="Tables"
isCollapsible={false}>
        <RibbonCollectionsDirective>
          <RibbonCollectionDirective>
            <RibbonItemsDirective>
              <RibbonItemDirective type="DropDown"
keyTip='T' allowedSizes={RibbonItemSize.Large} dropDownSettings={{ iconCss:
"e-icons e-table", items: tableOptions, content: "Table" }}>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        <RibbonGroupDirective header="Show"
isCollapsible={true}>
          <RibbonCollectionsDirective>
            <RibbonCollectionDirective>
              <RibbonItemsDirective>
                <RibbonItemDirective type="CheckBox"
keyTip='R1' checkBoxSettings={{ label: "Ruler", checked: false }}>
                  </RibbonItemDirective>
                <RibbonItemDirective type="CheckBox"
keyTip='R2' checkBoxSettings={{ label: "Gridlines", checked: false }}>
                  </RibbonItemDirective>
                <RibbonItemDirective type="CheckBox"
keyTip='R3' checkBoxSettings={{ label: "Navigation Pane", checked: true }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
    <Inject services={[RibbonGallery, RibbonColorPicker,
RibbonKeyTip]} />
  </RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel, RibbonItemSize,
RibbonGroupButtonSelection, RibbonColorPicker, RibbonKeyTip } from
'@syncfusion/ej2-react-ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {
  let ribbonObj = React.useRef<RibbonComponent>(null);
  const ribbonCreated = () => {
    ribbonObj.current?.ribbonKeyTipModule.showKeyTips('H');
  }
  const gallerySettings: RibbonGallerySettingsModel = (
    {
      itemCount: 3,
      groups: [{
        itemWidth: '100',
        header: 'Styles',
        items: [{
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {
          content: 'Heading 2'
        }, {
          content: 'Heading 3'
        }, {
          content: 'Heading 4'
        }, {
          content: 'Heading 5'
        }]
      }]
    }
  );

  const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "Draw
Table" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge Format" }, { text: "Keep Text Only" }];
  const fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
"20", "22", "24", "26", "28", "36", "48", "72", "96"];
  const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];

  return (
    <div>

```



```

    <RibbonComponent id='ribbon' ref={ribbonObj} enableKeyTips={true}
    created={ribbonCreated}>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home' keyTip='H'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
            id="clipboard" keyTip='CD' groupIconCss="e-icons e-paste"
            showLauncherIcon={true}>
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
                    type="SplitButton" keyTip='PA' id="pastebtn"
                    allowedSizes={RibbonItemSize.Large}
                    splitButtonSettings={{ iconCss:
                    "e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            <RibbonCollectionDirective>
              <RibbonItemsDirective>
                <RibbonItemDirective type="Button"
                keyTip='CU' buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
              </RibbonItemDirective>
                <RibbonItemDirective type="Button"
                keyTip='CO' buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
              </RibbonItemDirective>
                <RibbonItemDirective type="Button"
                keyTip='CS' buttonSettings={{ iconCss: "e-icons e-format-painter", content:
                "Format Painter" }}>
              </RibbonItemDirective>
            </RibbonItemsDirective>
          </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
      </RibbonGroupDirective>
      <RibbonGroupDirective header="Font" keyTip='FB'
      overflowHeader="More Font Options" groupIconCss="e-icons e-bold"
      isCollapsible={false} enableGroupOverflow={true} orientation="Row"
      cssClass='font-group'>
        <RibbonCollectionsDirective>
          <RibbonCollectionDirective>
            <RibbonItemsDirective>
              <RibbonItemDirective type="ComboBox"
              keyTip='O1' comboBoxSettings={{ dataSource: fontStyle, index: 3, label: 'Font
              Style', width: '115px', popupWidth: '150px', allowFiltering: true }}>
            </RibbonItemDirective>
              <RibbonItemDirective type="ComboBox"
              keyTip='O2' comboBoxSettings={{ dataSource: fontSize, index: 3, label: 'Font
              Size', width: '65px', popupWidth: '85px', allowFiltering: true }}>
            </RibbonItemDirective>
          </RibbonItemsDirective>
        </RibbonCollectionDirective>
      <RibbonCollectionDirective>
        <RibbonItemsDirective>
          <RibbonItemDirective
          type="GroupButton" keyTip='GB' allowedSizes={RibbonItemSize.Small}
          groupButtonSettings={{selection: RibbonGroupButtonSelection.Single, items:

```

```

[{{iconCss: 'e-icons e-bold', keyTip: '1', content: 'Bold', selected: true},
{iconCss: 'e-icons e-italic', keyTip: '2', content: 'Italic'}, {iconCss: 'e-
icons e-underline', keyTip: '3', content: 'Underline'}, {iconCss: 'e-icons e-
strikethrough', keyTip: '4', content: 'Strikethrough'}, {iconCss: 'e-icons e-
change-case', keyTip: '5', content: 'Change Case'}}}]>
    </RibbonItemDirective>
    </RibbonItemDirective>
type="ColorPicker" keyTip='CP' allowedSizes={RibbonItemSize.Small}
colorPickerSettings={{value: '#123456'}}>
    </RibbonItemDirective>
    </RibbonItemsDirective>
    </RibbonCollectionDirective>
    </RibbonCollectionsDirective>
</RibbonGroupDirective>
<RibbonGroupDirective header="Gallery">
    <RibbonCollectionsDirective>
        <RibbonCollectionDirective>
            <RibbonItemsDirective>
                <RibbonItemDirective type="Gallery"
keyTip='GY' gallerySettings={gallerySettings} >
                    </RibbonItemDirective>
                </RibbonItemsDirective>
            </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
    </RibbonGroupDirective>
    <RibbonGroupDirective header="Tables"
isCollapsible={false}>
        <RibbonCollectionsDirective>
            <RibbonCollectionDirective>
                <RibbonItemsDirective>
                    <RibbonItemDirective type="DropDown"
keyTip='T' allowedSizes={RibbonItemSize.Large} dropDownSettings={{ iconCss:
"e-icons e-table", items: tableOptions, content: "Table" }}>
                        </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        <RibbonGroupDirective header="Show"
isCollapsible={true}>
            <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                        <RibbonItemDirective type="CheckBox"
keyTip='R1' checkBoxSettings={{ label: "Ruler", checked: false }}>
                            </RibbonItemDirective>
                        <RibbonItemDirective type="CheckBox"
keyTip='R2' checkBoxSettings={{ label: "Gridlines", checked: false }}>
                            </RibbonItemDirective>
                        <RibbonItemDirective type="CheckBox"
keyTip='R3' checkBoxSettings={{ label: "Navigation Pane", checked: true }}>
                            </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
        </RibbonGroupDirective>
    </RibbonGroupsDirective>

```

```

        </RibbonTabDirective>
      </RibbonTabsDirective>
      <Inject services={[RibbonGallery, RibbonColorPicker,
RibbonKeyTip]} />
    </RibbonComponent>
  </div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

File menu keytip

You can add keytips to the file menu by using the [keyTip](#) property.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonItemSize, RibbonKeyTip, RibbonFileMenu } from '@syncfusion/ej2-react-ribbon';
function App() {
  let ribbonObj = React.useRef(null);
  const ribbonCreated = () => {
    ribbonObj.current?.ribbonKeyTipModule.showKeyTips();
  }

  const fileOptions = [
    { text: "New", iconCss: "e-icons e-file-new", id: "new" },
    { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
    { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
    { text: "Save as", iconCss: "e-icons e-save", id: "save" }
  ]

  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];

  return (
    <div>

```

```

        <RibbonComponent id='ribbon' ref={ribbonObj} enableKeyTips={true}
created={ribbonCreated} fileMenu={{ visible: true, keyTip: 'F', menuItems:
fileOptions}}>
            <RibbonTabsDirective>
                <RibbonTabDirective header='Home'>
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
id="clipboard" groupIconCss="e-icons e-paste" showLauncherIcon={true}>
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" id="pastebtn" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
                <Inject services={[RibbonFileMenu, RibbonKeyTip]} />
            </RibbonComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonItemSize, RibbonKeyTip, RibbonFileMenu } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
import { MenuItemModel } from '@syncfusion/ej2-navigations';
function App() {

```

```

let ribbonObj = React.useRef<RibbonComponent>(null);
const ribbonCreated = () => {
  ribbonObj.current?.ribbonKeyTipModule.showKeyTips();
}

const fileOptions: MenuItemModel[] = [
  { text: "New", iconCss: "e-icons e-file-new", id: "new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  { text: "Save as", iconCss: "e-icons e-save", id: "save" }
]

const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];

return (
  <div>
    <RibbonComponent id='ribbon' ref={ribbonObj} enableKeyTips={true}
created={ribbonCreated} fileMenu={{ visible: true, keyTip: 'F', menuItems:
fileOptions}}>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
id="clipboard" groupIconCss="e-icons e-paste" showLauncherIcon={true}>
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" id="pastebtn" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonFileMenu, RibbonKeyTip]} />
      </RibbonComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Backstage menu keytip

You can add keytips to backstage menu items by using the [keyTip](#) property.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonItemSize, RibbonKeyTip, RibbonBackstage } from '@syncfusion/ej2-react-ribbon';
function App() {
  let ribbonObj = React.useRef(null);
  const ribbonCreated = () => {
    ribbonObj.current?.ribbonKeyTipModule.showKeyTips('F');
  }
  const backstageSettings = {
    visible: true,
    keyTip: 'F',
    items: [
      { id: 'home', keyTip: 'H', text: 'Home', iconCss: 'e-icons e-home', content: homeContentTemplate() },
      { id: 'new', keyTip: 'N', text: 'New', iconCss: 'e-icons e-file-new', content: newContentTemplate() },
      { id: 'open', keyTip: 'O', text: 'Open', iconCss: 'e-icons e-folder-open', content: openContentTemplate() }
    ],
    backButton: {
      text: 'Close',
    }
  }
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge Format" }, { text: "Keep Text Only" }];
  function homeContentTemplate () {
    return "<div id='home-wrapper' style='padding: 20px;'><div id='new-section' class='new-wrapper'><div class='section-title'> New </div><div class='category_container'><div class='doc_category_image'></div><span class='doc_category_text'> New document </span></div></div><div id='block-wrapper'><div class='section-title'> Recent </div><div class='section-content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td><span class='doc_icon e-icons e-open-link'></span> </td><td><span

```

```

style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
    }

    function newContentTemplate () {
        return (
            "<div id='new-content' style='padding: 20px;'><div id='new-
            section' class='new-wrapper'><div class='section-title'> New </div><div
            class='category_container'><div class='doc_category_image'></div><span
            class='doc_category_text'> New document </span></div></div></div> "
        )
    }

    function openContentTemplate () {
        return (
            "<div style='padding: 20px;'><div class='section-content'
            style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
            class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
            block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
            12px'> Use the full functionality of Ribbon
            </span></td></tr></tbody></table></div><div class='section-content'
            style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
            class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
            style='display: block; font-size: 14px'> Protect Document </span><span
            style='font-size: 12px'>To prevent accidental changes, this document has been
            set to open as view-only.</span></td></tr></tbody></table></div></div>"
        )
    }

    return (
        <div>
            <RibbonComponent id='ribbon' ref={ribbonObj} enableKeyTips={true}
            created={ribbonCreated} backstageMenu={backstageSettings}>
                <RibbonTabsDirective>
                    <RibbonTabDirective header='Home'>
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Clipboard"
                            id="clipboard" groupIconCss="e-icons e-paste" showLauncherIcon={true}>
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
                                            type="SplitButton" id="pastebtn" allowedSizes={RibbonItemSize.Large}
                                            splitButtonSettings={{ iconCss:
                                            "e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective type="Button"
                                            buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective type="Button"
                                            buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                                </RibbonItemDirective>

```

```

        </RibbonItemsDirective>
        </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
        </RibbonGroupDirective>
        </RibbonGroupsDirective>
        </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonBackstage, RibbonKeyTip]} />
        </RibbonComponent>
    </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonItemSize, RibbonKeyTip, BackStageMenuModel, RibbonBackstage } from
"@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-splitbuttons";
function App() {

    let ribbonObj = React.useRef<RibbonComponent>(null);
    const ribbonCreated = () => {
        ribbonObj.current?.ribbonKeyTipModule.showKeyTips('F');
    }
    const backstageSettings: BackStageMenuModel = {
        visible: true,
        keyTip: 'F',
        items: [
            { id: 'home', keyTip: 'H', text: 'Home', iconCss: 'e-icons e-home',
content: homeContentTemplate() },
            { id: 'new', keyTip: 'N', text: 'New', iconCss: 'e-icons e-file-
new', content: newContentTemplate() },
            { id: 'open', keyTip: 'O', text: 'Open', iconCss: 'e-icons e-
folder-open', content: openContentTemplate() }
        ],
        backButton: {
            text: 'Close',
        }
    }
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge Format" }, { text: "Keep Text Only" }];
    function homeContentTemplate () {
        return "<div id='home-wrapper' style='padding: 20px;'><div id='new-
section' class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div><div id='block-
wrapper'><div class='section-title'> Recent </div><div class='section-

```



```

content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td>
<span class='doc_icon e-icons e-open-link'></span> </td><td><span
style='display: block; font-size: 14px'> Ribbon.docx </span><span
style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >>
default </span></td></tr></tbody></table></div></div></div>"
    }

    function newContentTemplate () {
        return (
            "<div id='new-content' style='padding: 20px;'><div id='new-section'
class='new-wrapper'><div class='section-title'> New </div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'> New document </span></div></div></div> "
        )
    }

    function openContentTemplate () {
        return (
            "<div style='padding: 20px;'><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-open-link'></span> </td><td><span style='display:
block; font-size: 14px'> Open in Desktop App </span><span style='font-size:
12px'> Use the full functionality of Ribbon
</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span
class='doc_icon e-icons e-protect-sheet'></span> </td><td><span
style='display: block; font-size: 14px'> Protect Document </span><span
style='font-size: 12px'>To prevent accidental changes, this document has been
set to open as view-only.</span></td></tr></tbody></table></div></div>"
        )
    }

    return (
        <div>
            <RibbonComponent id='ribbon' ref={ribbonObj} enableKeyTips={true}
created={ribbonCreated} backstageMenu={backstageSettings}>
                <RibbonTabsDirective>
                    <RibbonTabDirective header='Home'>
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Clipboard"
id="clipboard" groupIconCss="e-icons e-paste" showLauncherIcon={true}>
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="SplitButton" id="pastebtn" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                                </RibbonItemDirective>

```

```

                                <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonBackstage, RibbonKeyTip]} />
                                </RibbonComponent>
                                </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Ribbon layout switcher keytip

You can add keytip to the layout switcher by using the [layoutSwitcherKeyTip](#) property.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonItemSize, RibbonKeyTip, RibbonFileMenu } from '@syncfusion/ej2-react-ribbon';
function App() {
  let ribbonObj = React.useRef(null);
  const ribbonCreated = () => {
    ribbonObj.current?.ribbonKeyTipModule.showKeyTips();
  }

  const fileOptions = [
    { text: "New", iconCss: "e-icons e-file-new", id: "new" },
    { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
    { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
    { text: "Save as", iconCss: "e-icons e-save", id: "save" }
  ]
}

```

```

const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];

return (
  <div>
    <RibbonComponent id='ribbon' ref={ribbonObj} enableKeyTips={true}
layoutSwitcherKeyTip='LS' created={ribbonCreated} fileMenu={{ visible: true,
menuItems: fileOptions}}>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
id="clipboard" groupIconCss="e-icons e-paste">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" id="pastebtn" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonFileMenu, RibbonKeyTip]} />
      </RibbonComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,

```

```

RibbonItemSize, RibbonKeyTip, RibbonFileMenu } from '@syncfusion/ej2-react-ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
import { MenuItemModel } from '@syncfusion/ej2-navigations';
function App() {

    let ribbonObj = React.useRef<RibbonComponent>(null);
    const ribbonCreated = () => {
        ribbonObj.current?.ribbonKeyTipModule.showKeyTips();
    }

    const fileOptions: MenuItemModel[] = [
        { text: "New", iconCss: "e-icons e-file-new", id: "new" },
        { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
        { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
        { text: "Save as", iconCss: "e-icons e-save", id: "save" }
    ]
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];

    return (
        <div>
            <RibbonComponent id='ribbon' ref={ribbonObj} enableKeyTips={true}
layoutSwitcherKeyTip='LS' created={ribbonCreated} fileMenu={{ visible: true,
menuItems: fileOptions}}>
                <RibbonTabsDirective>
                    <RibbonTabDirective header='Home'>
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Clipboard"
id="clipboard" groupIconCss="e-icons e-paste">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="SplitButton" id="pastebtn" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                                </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
                <Inject services={[RibbonFileMenu, RibbonKeyTip]} />
            </RibbonComponent>
        </div>
    )
}

```

```

        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Ribbon launcher icon keytip

You can add keytip to the launcher icon by using the [launcherIconKeyTip](#) property.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonItemSize, RibbonKeyTip, RibbonFileMenu } from '@syncfusion/ej2-react-ribbon';
function App() {
  let ribbonObj = React.useRef(null);
  const ribbonCreated = () => {
    ribbonObj.current?.ribbonKeyTipModule.showKeyTips('H');
  }

  const fileOptions = [
    { text: "New", iconCss: "e-icons e-file-new", id: "new" },
    { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
    { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
    { text: "Save as", iconCss: "e-icons e-save", id: "save" }
  ]
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];

  return (
    <div>
      <RibbonComponent id='ribbon' ref={ribbonObj} enableKeyTips={true}
created={ribbonCreated} fileMenu={{ visible: true, menuItems: fileOptions}}>
        <RibbonTabsDirective>
          <RibbonTabDirective header='Home' keyTip='H'>
            <RibbonGroupsDirective>

```

```

        <RibbonGroupDirective header="Clipboard"
id="clipboard" groupIconCss="e-icons e-paste" showLauncherIcon={true}
launcherIconKeyTip='L'>
            <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                        <RibbonItemDirective
type="SplitButton" id="pastebtn" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                        </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
                <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                        </RibbonItemDirective>
                        <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                        </RibbonItemDirective>
                    </RibbonItemsDirective>
                </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
        </RibbonGroupDirective>
    </RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonFileMenu, RibbonKeyTip]} />
</RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonItemSize, RibbonKeyTip, RibbonFileMenu } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
import { MenuItemModel } from '@syncfusion/ej2-navigations';
function App() {

    let ribbonObj = React.useRef<RibbonComponent>(null);
    const ribbonCreated = () => {
        ribbonObj.current?.ribbonKeyTipModule.showKeyTips('H');
    }
}

```

```

const fileOptions: MenuItemModel[] = [
  { text: "New", iconCss: "e-icons e-file-new", id: "new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  { text: "Save as", iconCss: "e-icons e-save", id: "save" }
]

const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];

return (
  <div>
    <RibbonComponent id='ribbon' ref={ribbonObj} enableKeyTips={true}
created={ribbonCreated} fileMenu={{ visible: true, menuItems: fileOptions}}>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home' keyTip='H'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
id="clipboard" groupIconCss="e-icons e-paste" showLauncherIcon={true}
launcherIconKeyTip='L'>
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" id="pastebtn" allowedSizes={RibbonItemSize.Large}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOptions, content: "Paste" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                  <RibbonItemDirective type="Button"
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                      </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
      <Inject services={[RibbonFileMenu, RibbonKeyTip]} />
    </RibbonComponent>
  </div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```
/* Represents the styles for loader */
```

```
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
```

Methods

Show keytips

You can use the [showKeyTips](#) method to shown the keytips dynamically.

In order to show specific keytips, pass the key string as an argument in the `showKeyTips('H')` method.

Hide keytips

You can use the [hideKeyTips](#) method in Ribbon to remove the keytips dynamically. This will remove all the visible keytips.

Guidelines for adding keytips

Before adding keytips to the ribbon items consider the following:

- Avoid using the same keytip setting on multiple items.

For example: When you add the keytip text `H` or `HF` for the same items, it activates the first item occurrence of `H`, while any subsequent instances of `H` or `HF` are ignored.

- Do not use the same first letter for the single and double keytip items.

For example: When accessing keytip text `F`, `FP` and `FPF` added for the different ribbon items and pressing `F` key, only the `F` key tip associated item will be activated while the `FP`, `FPF` configured ribbon items will be ignored.

Gallery Items in React Ribbon component

The Ribbon supports Gallery view which allows users to perform specific actions by displaying a collection of related items, including icons, content, or images. You can render the gallery Ribbon items by using the `RibbonItemDirective` tag, by specifying the [type](#) property to `Gallery` and customize it by using the [RibbonGallerySettingsModel](#), which provides options such as `groups`, `itemCount`, `popupHeight`, `popupWidth` and more.

Groups

You can render the groups inside the gallery items by using [groups](#) property and customize the groups using [RibbonGalleryGroupModel](#), which provides options such as `items`, `cssClass`, `header` and more.

Adding items

You can render the gallery items by using [items](#) property and customize using the [RibbonGalleryItemModel](#), which provides options such as `content`, `iconCss`, `disabled` and more.

INDEX.JSX

```
{% raw %}
```



```

import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {
    const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
    const gallerySettings = (
        {
            groups: [{
                header: 'Styles',
                items: [
                    {
                        content: 'Normal'
                    }, {
                        content: 'No Spacing'
                    }, {
                        content: 'Heading 1'
                    }, {
                        content: 'Heading 2'
                    }
                ]
            }]
        }
    );
    return (
        <div>
            <RibbonComponent id='ribbon'>
                <RibbonTabsDirective>
                    <RibbonTabDirective header='Home'>
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                                </RibbonItemDirective>

```

```

        </RibbonItemsDirective>
      </RibbonCollectionDirective>
    </RibbonCollectionsDirective>
  </RibbonGroupDirective>
  <RibbonGroupDirective header="Gallery">
    <RibbonCollectionsDirective>
      <RibbonCollectionDirective>
        <RibbonItemsDirective>
          <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
            </RibbonItemDirective>
          </RibbonItemsDirective>
        </RibbonCollectionDirective>
      </RibbonCollectionsDirective>
    </RibbonGroupDirective>
  </RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonGallery]} />
</RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {

  const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
  const gallerySettings: RibbonGallerySettingsModel = (
    {
      groups: [{
        header: 'Styles',
        items: [
          {
            content: 'Normal'
          }, {
            content: 'No Spacing'
          }, {
            content: 'Heading 1'
          }, {
            content: 'Heading 2'
          }
        ]
      }
    ]
  );
}

```

```

    ]
  }
}
);
return (
  <div>
    <RibbonComponent id='ribbon'>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                      </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            <RibbonGroupDirective header="Gallery">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonGallery]} />
      </RibbonComponent>
    </div>
  );
}

```

```
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}
```

Adding content

You can use the [content](#) property to define the text content for the gallery item.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
  const gallerySettings = (
    {
      groups: [{
        header: 'Styles',
        items: [
          {
            content: 'Normal'
          }, {
            content: 'No Spacing'
          }, {
            content: 'Heading 1'
          }, {
            content: 'Heading 2'
          }
        ]
      }
    ]
  );
  return (
    <div>
      <RibbonComponent id='ribbon'>
        <RibbonTabsDirective>
          <RibbonTabDirective header='Home'>
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionDirective>
            </RibbonItemsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
    </div>
  );
}
```

```

                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                </RibbonItemDirective>
                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Gallery">
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonGallery]} />
                                </RibbonComponent>
                                </div>
                                );
                                }
                                export default App;
                                ReactDOM.render(<App />, document.getElementById("element"));
                                {% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {

    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
    const gallerySettings: RibbonGallerySettingsModel = (
        {
            groups: [{

```

```

        header: 'Styles',
        items: [
            {
                content: 'Normal'
            }, {
                content: 'No Spacing'
            }, {
                content: 'Heading 1'
            }, {
                content: 'Heading 2'
            }
        ]
    }
}

);
return (
    <div>
        <RibbonComponent id='ribbon'>
            <RibbonTabsDirective>
                <RibbonTabDirective header='Home'>
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        <RibbonGroupDirective header="Gallery">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
            </RibbonComponent>
        </div>
    </return>

```

```

        </RibbonCollectionDirective>
      </RibbonCollectionsDirective>
    </RibbonGroupDirective>
  </RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonGallery]} />
</RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

Adding icons

You can use the [iconCss](#) property to define the icons for the gallery item.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
  const gallerySettings = (
    {
      groups: [{
        header: 'Transitions',
        items: [
          {
            content: 'None',
            iconCss: 'e-icons e-rectangle'
          }, {
            content: 'Fade',
            iconCss: 'e-icons e-send-backward'
          }, {
            content: 'Reveal',
            iconCss: 'e-icons e-bring-forward'
          }, {
            content: 'Zoom',
            iconCss: 'e-icons e-zoom-to-fit'
          }
        ]
      }
    ]
  )
};
return (
  <div>
    <RibbonComponent id='ribbon'>
      <RibbonTabsDirective>

```

```

        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            <RibbonGroupDirective header="Gallery">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonGallery]} />
      </RibbonComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```
{% raw %}
```



```

import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {

    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
    const gallerySettings: RibbonGallerySettingsModel = (
        {
            groups: [{
                header: 'Transitions',
                items: [
                    {
                        content: 'None',
                        iconCss: 'e-icons e-rectangle'
                    }, {
                        content: 'Fade',
                        iconCss: 'e-icons e-send-backward'
                    }, {
                        content: 'Reveal',
                        iconCss: 'e-icons e-bring-forward'
                    }, {
                        content: 'Zoom',
                        iconCss: 'e-icons e-zoom-to-fit'
                    }
                ]
            }
        ]
    );

    return (
        <div>
            <RibbonComponent id='ribbon'>
                <RibbonTabsDirective>
                    <RibbonTabDirective header='Home'>
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                                </RibbonItemDirective>

```

```

                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                </RibbonItemDirective>
                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Gallery">
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonGallery]} />
                                </RibbonComponent>
                                </div>
                                );
                                }
                                export default App;
                                ReactDOM.render(<App />, document.getElementById("element"));
                                {% endraw %}

```

Adding html attributes

You can use the [htmlAttributes](#) property to add HTML attributes to the Ribbon gallery item.

The following sample showcases how to add title attribute to the gallery item using `htmlAttributes` property.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {
    const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
    const gallerySettings = (
        {
            groups: [{

```

```

        header: 'Styles',
        items: [
            {
                content: 'Normal',
                htmlAttributes: { title: "Normal" }
            }, {
                content: 'No Spacing',
                htmlAttributes: { title: "No Spacing" }
            }, {
                content: 'Heading 1',
                htmlAttributes: { title: "Heading 1" }
            }, {
                content: 'Heading 2',
                htmlAttributes: { title: "Heading 2" }
            }
        ]
    }
}

);
return (
    <div>
        <RibbonComponent id='ribbon'>
            <RibbonTabsDirective>
                <RibbonTabDirective header='Home'>
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        <RibbonGroupDirective header="Gallery">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>

```

```

                                <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonGallery]} />
                                </RibbonComponent>
                                </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {

    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
    const gallerySettings: RibbonGallerySettingsModel = (
        {
            groups: [{
                header: 'Styles',
                items: [
                    {
                        content: 'Normal',
                        htmlAttributes: { title: "Normal" }
                    }, {
                        content: 'No Spacing',
                        htmlAttributes: { title: "No Spacing" }
                    }, {
                        content: 'Heading 1',
                        htmlAttributes: { title: "Heading 1" }
                    }, {
                        content: 'Heading 2',
                        htmlAttributes: { title: "Heading 2" }
                    }
                ]
            }
        ]
    )
}
};

```

```

    return (
      <div>
        <RibbonComponent id='ribbon'>
          <RibbonTabsDirective>
            <RibbonTabDirective header='Home'>
              <RibbonGroupsDirective>
                <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                  <RibbonCollectionsDirective>
                    <RibbonCollectionDirective>
                      <RibbonItemsDirective>
                        <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                        </RibbonItemDirective>
                      </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                      </RibbonItemDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
              <RibbonGroupDirective header="Gallery">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonGallery]} />
      </RibbonComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% enddraw %}

```

Css class

You can use the [cssClass](#) property to customize the gallery item.

The following sample showcases how to customize the appearance of each gallery item using the `cssClass` property .

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {
    const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
    const gallerySettings = (
        {
            groups: [{
                header: 'Good, Bad and Neutral',
                items: [{
                    content: 'Normal',
                    cssClass: 'normal'
                }, {
                    content: 'Bad',
                    cssClass: 'bad'
                }, {
                    content: 'Good',
                    cssClass: 'good'
                }, {
                    content: 'Neutral',
                    cssClass: 'neutral'
                }
            ]
        }
    ]);
    return (
        <div>
            <RibbonComponent id='ribbon'>
                <RibbonTabsDirective>
                    <RibbonTabDirective header='Home'>
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                            </RibbonGroupsDirective>
                        </RibbonTabDirective>
                    </RibbonTabsDirective>
                </RibbonComponent>
            </div>
        );
    }
}
```

```

                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                        </RibbonItemDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                        <RibbonGroupDirective header="Gallery">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
            <Inject services={[RibbonGallery]} />
        </RibbonComponent>
    </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {
    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
    const gallerySettings: RibbonGallerySettingsModel = (

```

```

    {
      groups: [{
        header: 'Good, Bad and Neutral',
        items: [{
          content: 'Normal',
          cssClass: 'normal'
        }, {
          content: 'Bad',
          cssClass: 'bad'
        }, {
          content: 'Good',
          cssClass: 'good'
        }, {
          content: 'Neutral',
          cssClass: 'neutral'
        }
      ]
    }
  ]
};
return (
  <div>
    <RibbonComponent id='ribbon'>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                      </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            <RibbonGroupDirective header="Gallery">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>

```



```

                                <RibbonItemsDirective>
                                    <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        <Inject services={[RibbonGallery]} />
    </RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.e-ribbon-gallery-item {
  margin: 5px;
}
.e-ribbon-gallery-item.normal{
  background: #f0f0f0;
  color: #333;
}
.e-ribbon-gallery-item.bad {
  background: #ffb6b6;
  color: #800000;
}
.e-ribbon-gallery-item.good {
  background: #c7ebc9;
  color: #004d00;
}
.e-ribbon-gallery-item.neutral {
  background: #eedd9d;
  color: #6c5429;
}

```

Disabled

You can use the [disabled](#) property to disable the Ribbon gallery item. It prevents the user interaction when set to `true`. By default, the value is `false`.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {
    const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
    const gallerySettings = (
        {
            groups: [{
                header: 'Styles',
                items: [
                    {
                        content: 'Normal',
                        disabled: true
                    }, {
                        content: 'No Spacing'
                    }, {
                        content: 'Heading 1'
                    }, {
                        content: 'Heading 2'
                    }
                ]
            }
        ]
    );
    return (
        <div>
            <RibbonComponent id='ribbon'>
                <RibbonTabsDirective>
                    <RibbonTabDirective header='Home'>
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                                </RibbonItemDirective>

```

```

                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Gallery">
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonGallery]} />
                                </RibbonComponent>
                                </div>
                                );
                                }
                                export default App;
                                ReactDOM.render(<App />, document.getElementById("element"));
                                {% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {

    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
    const gallerySettings: RibbonGallerySettingsModel = (
        {
            groups: [{
                header: 'Styles',
                items: [
                    {
                        content: 'Normal',
                        disabled: true
                    }, {
                        content: 'No Spacing'

```

```

        }, {
            content: 'Heading 1'
        }, {
            content: 'Heading 2'
        }
    ]
}
}
];
};
return (
    <div>
        <RibbonComponent id='ribbon'>
            <RibbonTabsDirective>
                <RibbonTabDirective header='Home'>
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            <RibbonGroupDirective header="Gallery">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                            </RibbonGroupsDirective>
                        </RibbonTabDirective>
                    </RibbonTabsDirective>
                </RibbonGroupDirective>
            </RibbonGroupsDirective>
        </RibbonComponent>
    </div>
);

```

```

        <Inject services={[RibbonGallery]} />
      </RibbonComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

Custom header

You can use the [header](#) property to set header for the group items in the Ribbon gallery popup.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
  const gallerySettings = (
    {
      groups: [{
        header: 'Styles',
        items: [
          {
            content: 'Normal'
          }, {
            content: 'No Spacing'
          }, {
            content: 'Heading 1'
          }, {
            content: 'Heading 2'
          }
        ]
      }
    ]
  );
  return (
    <div>
      <RibbonComponent id='ribbon'>
        <RibbonTabsDirective>
          <RibbonTabDirective header='Home'>
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>

```

```

        </RibbonItemDirective>
      </RibbonItemsDirective>
    </RibbonCollectionDirective>
  </RibbonCollectionDirective>
    <RibbonItemsDirective>
      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
      </RibbonItemDirective>
      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
      </RibbonItemDirective>
      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
      </RibbonItemDirective>
    </RibbonItemsDirective>
  </RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
<RibbonGroupDirective header="Gallery">
  <RibbonCollectionsDirective>
    <RibbonCollectionDirective>
      <RibbonItemsDirective>
        <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
        </RibbonItemDirective>
      </RibbonItemsDirective>
    </RibbonCollectionDirective>
  </RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonGallery]} />
</RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {

```

```

const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
const gallerySettings: RibbonGallerySettingsModel = (
  {
    groups: [{
      header: 'Styles',
      items: [
        {
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {
          content: 'Heading 2'
        }
      ]
    }
  ]
});
return (
  <div>
    <RibbonComponent id='ribbon'>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                      </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            <RibbonGroupDirective header="Gallery">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>

```

```

                                <RibbonItemsDirective>
                                    <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
        <Inject services={[RibbonGallery]} />
    </RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
[% endraw %]

```

Setting item width

You can use the [itemWidth](#) property to specify the width of gallery items.

Setting item height

You can use the [itemHeight](#) property to set the height of the gallery items. If the [itemHeight](#) of the gallery item is smaller the remaining gallery items are aligned based on the [itemCount](#) specified.

The provided example demonstrates how to customize gallery items using the [itemWidth](#) and [itemHeight](#) properties.

INDEX.JSX

```

[% raw %]
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {
    const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
    const gallerySettings = (
        {
            popupWidth: '350',
            groups: [{
                itemWidth: '100',
                itemHeight: '30',
                header: 'Styles',
                items: [
                    {
                        content: 'Normal'
                    }, {
                        content: 'No Spacing'
                    }, {
                        content: 'Heading 1'
                    }
                ]
            }
        ]
    );
}

```



```

        }, {
            content: 'Heading 2'
        }, {
            content: 'Title'
        }, {
            content: 'Subtitle'
        }
    ]
}
}
];
);
return (
    <div>
        <RibbonComponent id='ribbon'>
            <RibbonTabsDirective>
                <RibbonTabDirective header='Home'>
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        <RibbonGroupDirective header="Gallery">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonGroupDirective>
                </RibbonGroupsDirective>
            </RibbonTabDirective>
        </RibbonTabsDirective>
    </RibbonComponent>
    </div>
);

```

```

        </RibbonTabDirective>
      </RibbonTabsDirective>
      <Inject services={[RibbonGallery]} />
    </RibbonComponent>
  </div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% enddraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {

  const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
  const gallerySettings: RibbonGallerySettingsModel = (
    {
      popupWidth: '350',
      groups: [{
        itemWidth: '100',
        itemHeight: '30',
        header: 'Styles',
        items: [
          {
            content: 'Normal'
          }, {
            content: 'No Spacing'
          }, {
            content: 'Heading 1'
          }, {
            content: 'Heading 2'
          }, {
            content: 'Title'
          }, {
            content: 'Subtitle'
          }
        ]
      }]
    }
  );
  return (
    <div>
      <RibbonComponent id='ribbon'>
        <RibbonTabsDirective>
          <RibbonTabDirective header='Home'>

```

```

        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionDirective>
            <RibbonItemsDirective>
              <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
              </RibbonItemDirective>
              <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
              </RibbonItemDirective>
              <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
              </RibbonItemDirective>
            </RibbonItemsDirective>
          </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
      </RibbonGroupDirective>
    </RibbonGroupDirective header="Gallery">
      <RibbonCollectionsDirective>
        <RibbonCollectionDirective>
          <RibbonItemsDirective>
            <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
            </RibbonItemDirective>
          </RibbonItemsDirective>
        </RibbonCollectionDirective>
      </RibbonCollectionsDirective>
    </RibbonGroupDirective>
  </RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonGallery]} />
</RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

Group styling

You can use the [cssClass](#) property to customize the appearance of gallery groups.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
  const gallerySettings = (
    {
      groups: [{
        header: 'Styles',
        cssClass: "custom-group",
        items: [
          {
            content: 'Normal'
          }, {
            content: 'No Spacing'
          }, {
            content: 'Heading 1'
          }, {
            content: 'Heading 2'
          }
        ]
      }
    ]
  );
  return (
    <div>
      <RibbonComponent id='ribbon'>
        <RibbonTabsDirective>
          <RibbonTabDirective header='Home'>
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                        </RibbonItemDirective>
                      </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                        </RibbonItemDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                        </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonGroupsDirective>
              </RibbonTabDirective>
            </RibbonTabsDirective>
          </RibbonComponent>
        </div>
      </App>
    </pre>

```

```

                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Gallery">
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonGallery]} />
                                </RibbonComponent>
                                </div>
                                );
                                }
                                export default App;
                                ReactDOM.render(<App />, document.getElementById("element"));
                                {% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {

    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
    const gallerySettings: RibbonGallerySettingsModel = (
        {
            groups: [{
                header: 'Styles',
                cssClass: "custom-group",
                items: [
                    {
                        content: 'Normal'
                    }, {
                        content: 'No Spacing'

```

```

        }, {
            content: 'Heading 1'
        }, {
            content: 'Heading 2'
        }
    ]
}
}
];
};
return (
    <div>
        <RibbonComponent id='ribbon'>
            <RibbonTabsDirective>
                <RibbonTabDirective header='Home'>
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            <RibbonGroupDirective header="Gallery">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                            </RibbonGroupsDirective>
                        </RibbonTabDirective>
                    </RibbonTabsDirective>
                </RibbonGroupDirective>
            </RibbonGroupsDirective>
        </RibbonComponent>
    </div>
);

```

```

        <Inject services={[RibbonGallery]} />
      </RibbonComponent>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.custom-group {
  font-style: italic;
}

```

Setting item count

You can customize the number of items to be displayed in Ribbon gallery by using the [itemCount](#) property. By default the `itemCount` will be 3.

The following example showcases the utilization of the `itemCount` property, displaying a ribbon gallery with 4 items.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
  const gallerySettings = (
    {
      itemCount: 4,
      groups: [{
        header: 'Styles',
        items: [
          {
            content: 'Normal'
          }, {
            content: 'No Spacing'
          }, {
            content: 'Heading 1'

```

```

        }, {
            content: 'Heading 2'
        }
    ]
}
}
];
};
return (
    <div>
        <RibbonComponent id='ribbon'>
            <RibbonTabsDirective>
                <RibbonTabDirective header='Home'>
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                            </RibbonItemDirective>
                                        <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        <RibbonGroupDirective header="Gallery">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
                <Inject services={[RibbonGallery]} />
            </RibbonComponent>
        </div>
    </return>

```



```

        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {

    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
    const gallerySettings: RibbonGallerySettingsModel = (
        {
            itemCount: 4,
            groups: [{
                header: 'Styles',
                items: [
                    {
                        content: 'Normal'
                    }, {
                        content: 'No Spacing'
                    }, {
                        content: 'Heading 1'
                    }, {
                        content: 'Heading 2'
                    }
                ]
            }]
        }
    );
    return (
        <div>
            <RibbonComponent id='ribbon'>
                <RibbonTabsDirective>
                    <RibbonTabDirective header='Home'>
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                                </RibbonItemDirective>

```

```

        </RibbonItemsDirective>
      </RibbonCollectionDirective>
    </RibbonCollectionDirective>
    <RibbonItemsDirective>
      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
      </RibbonItemDirective>
      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
    }}>
      </RibbonItemDirective>
      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
      </RibbonItemDirective>
    </RibbonItemsDirective>
  </RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
<RibbonGroupDirective header="Gallery">
  <RibbonCollectionsDirective>
    <RibbonCollectionDirective>
      <RibbonItemsDirective>
        <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
        </RibbonItemDirective>
      </RibbonItemsDirective>
    </RibbonCollectionDirective>
  </RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonGallery]} />
</RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

Setting selected item

You can use the [selectedItemIndex](#) property to define the currently selected item in the Ribbon gallery items.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {

```

```

const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
const gallerySettings = (
  {
    selectedItemIndex: 1,
    groups: [{
      header: 'Styles',
      items: [
        {
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {
          content: 'Heading 2'
        }
      ]
    }
  ]
});
return (
  <div>
    <RibbonComponent id='ribbon'>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                      </RibbonItemDirective>
                    <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            <RibbonGroupDirective header="Gallery">
              <RibbonCollectionsDirective>

```

```

                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                        </RibbonItemDirective>
                                    </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                            </RibbonCollectionsDirective>
                        </RibbonGroupDirective>
                    </RibbonGroupsDirective>
                </RibbonTabDirective>
            </RibbonTabsDirective>
            <Inject services={[RibbonGallery]} />
        </RibbonComponent>
    </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-
ribbon';
import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {

    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
    const gallerySettings: RibbonGallerySettingsModel = (
        {
            selectedItemIndex: 1,
            groups: [{
                header: 'Styles',
                items: [
                    {
                        content: 'Normal'
                    }, {
                        content: 'No Spacing'
                    }, {
                        content: 'Heading 1'
                    }, {
                        content: 'Heading 2'
                    }
                ]
            }]
        }
    );
    return (

```

```

    <div>
      <RibbonComponent id='ribbon'>
        <RibbonTabsDirective>
          <RibbonTabDirective header='Home'>
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                        </RibbonItemDirective>
                      </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                        </RibbonItemDirective>
                      <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                        </RibbonItemDirective>
                      </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                  </RibbonCollectionsDirective>
                </RibbonGroupDirective>
              <RibbonGroupDirective header="Gallery">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                        </RibbonItemDirective>
                      </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                  </RibbonCollectionsDirective>
                </RibbonGroupDirective>
              </RibbonGroupsDirective>
            </RibbonTabDirective>
          </RibbonTabsDirective>
          <Inject services={[RibbonGallery]} />
        </RibbonComponent>
      </div>
    );
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById("element"));
  {% endraw %}

```

Setting popup height

You can specify the height of the gallery popup by using the [popupHeight](#) property.

Setting popup width

you can specify the width of the gallery popup by using the [popupWidth](#) property.

The example demonstrates the customization of popup with `popupHeight` and `popupWidth` properties.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery } from '@syncfusion/ej2-react-ribbon';
function App() {
  const pasteOptions = [{ text: "Keep Source Format" }, { text: "Merge
Format" }, { text: "Keep Text Only" }];
  const gallerySettings = (
    {
      popupHeight: '180',
      popupWidth: '350',
      groups: [{
        header: 'Styles',
        items: [
          {
            content: 'Normal'
          }, {
            content: 'No Spacing'
          }, {
            content: 'Heading 1'
          }, {
            content: 'Heading 2'
          }, {
            content: 'Title'
          }, {
            content: 'Subtitle'
          }
        ]
      }]
    }
  );
  return (
    <div>
      <RibbonComponent id='ribbon'>
        <RibbonTabsDirective>
          <RibbonTabDirective header='Home'>
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
```

```

                                <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                </RibbonItemDirective>
                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Gallery">
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonGallery]} />
                                </RibbonComponent>
                                </div>
                                );
    }
    export default App;
    ReactDOM.render(<App />, document.getElementById("element"));
    {% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective, Inject,
RibbonGallery, RibbonGallerySettingsModel } from '@syncfusion/ej2-react-
ribbon';

```

```

import { ItemModel } from '@syncfusion/ej2-splitbuttons';
function App() {

    const pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge Format" }, { text: "Keep Text Only" }];
    const gallerySettings: RibbonGallerySettingsModel = (
        {
            popupHeight: '180',
            popupWidth: '350',
            groups: [{
                header: 'Styles',
                items: [
                    {
                        content: 'Normal'
                    }, {
                        content: 'No Spacing'
                    }, {
                        content: 'Heading 1'
                    }, {
                        content: 'Heading 2'
                    }, {
                        content: 'Title'
                    }, {
                        content: 'Subtitle'
                    }
                ]
            }]
        }
    );
    return (
        <div>
            <RibbonComponent id='ribbon'>
                <RibbonTabsDirective>
                    <RibbonTabDirective header='Home'>
                        <RibbonGroupsDirective>
                            <RibbonGroupDirective header="Clipboard"
groupIconCss="e-icons e-paste">
                                <RibbonCollectionsDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-paste", items:
pasteOptions, content: "Paste" }}>
                                                </RibbonItemDirective>
                                            </RibbonItemsDirective>
                                        </RibbonCollectionDirective>
                                    <RibbonCollectionDirective>
                                        <RibbonItemsDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                                                </RibbonItemDirective>
                                            <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy"
}}>
                                                </RibbonItemDirective>

```



```

                                <RibbonItemDirective
type="Button" buttonSettings={{ iconCss: "e-icons e-format-painter", content:
"Format Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                <RibbonGroupDirective header="Gallery">
                                <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                <RibbonItemsDirective>
                                <RibbonItemDirective
type="Gallery" gallerySettings={gallerySettings} >
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                <Inject services={[RibbonGallery]} />
                                </RibbonComponent>
                                </div>
                                );
                                }
                                export default App;
                                ReactDOM.render(<App />, document.getElementById("element"));
                                {% endraw %}

```

To know more about the built-in Ribbon items, please refer to the [Ribbon Items](#) section.

Help Pane

The help pane is dedicated area where the users can define help contents like controlling document permissions, sharing features, and more which appears on the right side of the Ribbon. You can use the [helpPaneTemplate](#) property to set the help pane contents.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
    function helpPaneTemplate() {
        return (
            <React.Fragment>
                <button className="action_btn"><span id="undo" className="e-
icons e-undo"></span> Undo </button>
                <button className="action_btn"><span id="redo" className="e-
icons e-redo"></span> Redo </button>
            </React.Fragment>
        );
    }
}

```

```

    );
  }
  return (
    <RibbonComponent id="ribbon" helpPaneTemplate={helpPaneTemplate}>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Large} buttonSettings={{ iconCss: "e-icons e-
cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
      </RibbonComponent>
    );
  }
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  function helpPaneTemplate() {
    return (
      <React.Fragment>
        <button className="action_btn"><span id="undo" className="e-
icons e-undo"></span> Undo </button>
        <button className="action_btn"><span id="redo" className="e-
icons e-redo"></span> Redo </button>
      </React.Fragment>
    );
  }
  return (
    <RibbonComponent id="ribbon" helpPaneTemplate={helpPaneTemplate}>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>

```

```

        <RibbonCollectionDirective>
            <RibbonItemsDirective>
                <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Large} buttonSettings={{ iconCss: "e-icons e-
cut", content: "Cut" }}>
                    </RibbonItemDirective>
                </RibbonItemsDirective>
            </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
    </RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.action_btn {
    margin: 0px 5px;
    border: none;
    color: #ffffff;
    background-color: #0d6efd;
}

```

Tooltip

The Ribbon component supports tooltip to show additional information in the Ribbon items. The tooltip appears when the user hovers over a Ribbon item.

Adding Title

You can use the [title](#) property to set the tooltip title for each Ribbon item.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {

```

```

const cutOptions = { title: "Cut" };
const copyOptions = { title: "Copy" };
const pasteOptions = { title: "Paste" };
const painterOptions = { title: "Format Painter" };
const pasteOption = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
return (
  <RibbonComponent id="ribbon">
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home">
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Clipboard">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
ribbonTooltipSettings={pasteOptions}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOption, content: "Paste" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";

```

```

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize, RibbonTooltipModel } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
  const cutOptions: RibbonTooltipModel = { title: "Cut" };
  const copyOptions: RibbonTooltipModel = { title: "Copy" };
  const pasteOptions: RibbonTooltipModel = { title: "Paste" };
  const painterOptions: RibbonTooltipModel = { title: "Format Painter" };
  const pasteOption: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge format" }, { text: "Keep text only" }];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="SplitButton"
allowedSizes={RibbonItemSize.Large} ribbonTooltipSettings={pasteOptions}
splitButtonSettings={{ iconCss: "e-icons e-paste",
items: pasteOption, content: "Paste" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionDirective>
            <RibbonItemsDirective>
              <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Large} ribbonTooltipSettings={cutOptions}
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
              </RibbonItemDirective>
              <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} ribbonTooltipSettings={copyOptions}
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
              </RibbonItemDirective>
              <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} ribbonTooltipSettings={painterOptions}
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
              </RibbonItemDirective>
            </RibbonItemsDirective>
          </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
      </RibbonGroupDirective>
    </RibbonGroupsDirective>
  </RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Adding Content

You can use the [content](#) property to set the tooltip content for each Ribbon item.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  const cutOptions = { title: "Cut", content: "Places the selected text or
object on the clipboard so that you can paste it somewhere else." };
  const copyOptions = { title: "Copy", content: "Copies the chosen text or
object to the clipboard so that you can reuse it elsewhere." };
  const pasteOptions = { title: "Paste", content: "Insert the clipboard
content where the cursor is currently placed." };
  const painterOptions = { title: "Format Painter", content: "Copies the
formatting style of a selected text or object and applies it to other content
within the document." };
  const pasteOption = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
ribbonTooltipSettings={pasteOptions}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOption, content: "Paste" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionDirective>
          </RibbonItemsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>
</RibbonComponent>

```

```

                                <RibbonItemDirective type="Button"
ribbonTooltipSettings={cutOptions} buttonSettings={{ iconCss: "e-icons e-
cut", content: "Cut" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
ribbonTooltipSettings={copyOptions} allowedSizes={RibbonItemSize.Medium}
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                                </RibbonItemDirective>
                                <RibbonItemDirective type="Button"
ribbonTooltipSettings={painterOptions} buttonSettings={{ iconCss: "e-icons e-
format-painter", content: "Format Painter" }}>
                                </RibbonItemDirective>
                                </RibbonItemsDirective>
                                </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                                </RibbonGroupDirective>
                                </RibbonGroupsDirective>
                                </RibbonTabDirective>
                                </RibbonTabsDirective>
                                </RibbonComponent>
        );
    }
    export default App;
    ReactDOM.render(<App />, document.getElementById("element"));
    {% enddraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize, RibbonTooltipModel } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
    const cutOptions: RibbonTooltipModel = { title: "Cut", content: "Places the
selected text or object on the clipboard so that you can paste it somewhere
else." };
    const copyOptions: RibbonTooltipModel = { title: "Copy", content: "Copies
the chosen text or object to the clipboard so that you can reuse it
elsewhere." };
    const pasteOptions: RibbonTooltipModel = { title: "Paste", content: "Insert
the clipboard content where the cursor is currently placed." };
    const painterOptions: RibbonTooltipModel = { title: "Format Painter",
content: "Copies the formatting style of a selected text or object and
applies it to other content within the document." };
    const pasteOption: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge format" }, { text: "Keep text only" }];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">

```

```

        <RibbonCollectionsDirective>
          <RibbonCollectionDirective>
            <RibbonItemsDirective>
              <RibbonItemDirective type="SplitButton"
allowedSizes={RibbonItemSize.Large} ribbonTooltipSettings={pasteOptions}
              splitButtonSettings={{ iconCss: "e-icons e-paste",
items: pasteOption, content: "Paste" }}>
            </RibbonItemDirective>
          </RibbonItemsDirective>
        </RibbonCollectionDirective>
      </RibbonCollectionDirective>
      <RibbonCollectionDirective>
        <RibbonItemsDirective>
          <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Large} ribbonTooltipSettings={cutOptions}
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
        </RibbonItemDirective>
        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} ribbonTooltipSettings={copyOptions}
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
        </RibbonItemDirective>
        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} ribbonTooltipSettings={painterOptions}
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
        </RibbonItemDirective>
      </RibbonItemsDirective>
    </RibbonCollectionDirective>
  </RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% enddraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Adding Icon

You can use the [iconCss](#) property to specify the icons to be displayed in the tooltip.

INDEX.JSX


```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
    const cutOptions = { title: "Cut", content: "Places the selected text or
object on the clipboard so that you can paste it somewhere else.", iconCss:
    "e-icons e-cut" };
    const copyOptions = { title: "Copy", content: "Copies the chosen text or
object to the clipboard so that you can reuse it elsewhere.", iconCss: "e-
icons e-copy" };
    const pasteOptions = { title: "Paste", content: "Insert the clipboard
content where the cursor is currently placed.", iconCss: "e-icons e-paste" };
    const painterOptions = { title: "Format Painter", content: "Copies the
formatting style of a selected text or object and applies it to other content
within the document.", iconCss: "e-icons e-format-painter" };
    const pasteOption = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
ribbonTooltipSettings={pasteOptions}
                                        splitButtonSettings={{ iconCss:
    "e-icons e-paste", items: pasteOption, content: "Paste" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
            </RibbonComponent>
        );
    }
}
ReactDOM.render(App, document.getElementById("root"));

```

```

        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize, RibbonTooltipModel } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
  const cutOptions: RibbonTooltipModel = { title: "Cut", content: "Places the
selected text or object on the clipboard so that you can paste it somewhere
else.", iconCss: "e-icons e-cut" };
  const copyOptions: RibbonTooltipModel = { title: "Copy", content: "Copies
the chosen text or object to the clipboard so that you can reuse it
elsewhere.", iconCss: "e-icons e-copy" };
  const pasteOptions: RibbonTooltipModel = { title: "Paste", content: "Insert
the clipboard content where the cursor is currently placed.", iconCss: "e-
icons e-paste" };
  const painterOptions: RibbonTooltipModel = { title: "Format Painter",
content: "Copies the formatting style of a selected text or object and
applies it to other content within the document.", iconCss: "e-icons e-
format-painter" };
  const pasteOption: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge format" }, { text: "Keep text only" }];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="SplitButton"
allowedSizes={RibbonItemSize.Large} ribbonTooltipSettings={pasteOptions}
splitButtonSettings={{ iconCss: "e-icons e-paste",
items: pasteOption, content: "Paste" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Large} ribbonTooltipSettings={cutOptions}
buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>

```

```

        </RibbonItemDirective>
        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} ribbonTooltipSettings={copyOptions}
buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
        </RibbonItemDirective>
        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Medium} ribbonTooltipSettings={painterOptions}
buttonSettings={{ iconCss: "e-icons e-format-painter", content: "Format
Painter" }}>
        </RibbonItemDirective>
    </RibbonItemsDirective>
    </RibbonCollectionDirective>
    </RibbonCollectionsDirective>
    </RibbonGroupDirective>
    </RibbonGroupsDirective>
    </RibbonTabDirective>
    </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% enddraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}

```

Customization

You can use the [cssClass](#) property to customize the appearance of the tooltip with your own custom styles.

INDEX.JSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  const cutOptions = { title: "Cut", content: "Places the selected text or
object on the clipboard so that you can paste it somewhere else.", cssClass:
"custom-tooltip" };

```

```

    const copyOptions = { title: "Copy", content: "Copies the chosen text or
object to the clipboard so that you can reuse it elsewhere.", cssClass:
"custom-tooltip" };
    const pasteOptions = { title: "Paste", content: "Insert the clipboard
content where the cursor is currently placed.", cssClass: "custom-tooltip" };
    const painterOptions = { title: "Format Painter", content: "Copies the
formatting style of a selected text or object and applies it to other content
within the document.", cssClass: "custom-tooltip" };
    const pasteOption = [{ text: "Keep Source Format" }, { text: "Merge
format" }, { text: "Keep text only" }];
    return (
      <RibbonComponent id="ribbon">
        <RibbonTabsDirective>
          <RibbonTabDirective header="Home">
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Clipboard">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
                    <RibbonItemsDirective>
                      <RibbonItemDirective
type="SplitButton" allowedSizes={RibbonItemSize.Large}
ribbonTooltipSettings={pasteOptions}
splitButtonSettings={{ iconCss:
"e-icons e-paste", items: pasteOption, content: "Paste" }}>
                        </RibbonItemDirective>
                      </RibbonItemsDirective>
                    </RibbonCollectionDirective>
                  </RibbonCollectionsDirective>
                </RibbonGroupDirective>
              </RibbonGroupsDirective>
            </RibbonTabDirective>
          </RibbonTabsDirective>
        </RibbonComponent>
      );
    }
  export default App;
  ReactDOM.render(<App />, document.getElementById("element"));
  {% endraw %}

```

INDEX.TSX

```

{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize, RibbonTooltipModel } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
  const cutOptions: RibbonTooltipModel = { title: "Cut", content: "Places the
selected text or object on the clipboard so that you can paste it somewhere
else.", cssClass: "custom-tooltip" };
  const copyOptions: RibbonTooltipModel = { title: "Copy", content: "Copies
the chosen text or object to the clipboard so that you can reuse it
elsewhere.", cssClass: "custom-tooltip" };
  const pasteOptions: RibbonTooltipModel = { title: "Paste", content: "Insert
the clipboard content where the cursor is currently placed.", cssClass:
"custom-tooltip" };
  const painterOptions: RibbonTooltipModel = { title: "Format Painter",
content: "Copies the formatting style of a selected text or object and
applies it to other content within the document.", cssClass: "custom-tooltip"
};
  const pasteOption: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge format" }, { text: "Keep text only" }];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="SplitButton"
allowedSizes={RibbonItemSize.Large} ribbonTooltipSettings={pasteOptions}
splitButtonSettings={{ iconCss: "e-icons e-paste",
items: pasteOption, content: "Paste" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}

```

```

        </RibbonCollectionsDirective>
    </RibbonGroupDirective>
    </RibbonGroupsDirective>
    </RibbonTabDirective>
    </RibbonTabsDirective>
    </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
:root {
  --borderColor: rgb(72, 72, 72);
  --black: #000000;
}
/* To customize the appearance of the tooltip */
.custom-tooltip.e-ribbon-tooltip.e-popup {
  border: 2px solid var(--borderColor);
  border-radius: 5px;
  background: var(--black);
}
/* To customize the arrow of the tooltip */
.custom-tooltip.e-ribbon-tooltip .e-arrow-tip .e-arrow-tip-inner.e-tip-top,
.custom-tooltip.e-ribbon-tooltip .e-arrow-tip .e-arrow-tip-inner.e-tip-bottom
{
  color: var(--black);;
}
.custom-tooltip.e-ribbon-tooltip .e-arrow-tip-outer.e-tip-top {
  border-bottom: 8px solid var(--borderColor);
}
.custom-tooltip.e-ribbon-tooltip .e-arrow-tip-outer.e-tip-bottom {
  border-top: 8px solid var(--borderColor);;
}
/* To change the size of the tooltip title */
.custom-tooltip.e-ribbon-tooltip .e-tip-content .e-ribbon-tooltip-title {
  font-size: 14px;
}
/* To change the size of the tooltip content */
.custom-tooltip.e-ribbon-tooltip .e-tip-content .e-ribbon-text-container .e-
ribbon-tooltip-content {
  font-size: 11px;
}

```

Ribbon Resizing

The Ribbon effectively resizes the ribbon elements while being resized. It extends when the ribbon size is increased and collapses when the ribbon size is decreased. The resizing can be performed in both the classic and simplified modes. Also, we have an option to resize the ribbon elements in the custom order.

In classic mode on resizing, the items size will be changed based on the available width of the tab content from the order of Large-> Medium-> Small and viceversa.

In simplified mode on resizing, the items size will be changed based on the available width of the tab content from the order of Medium-> Small and viceversa.

Defining items allowed size

You can use the [allowedSizes](#) property to maintain a constant size for an item. If `allowedSizes` is set, it keeps the size constant even when being resized.

INDEX.JSX

```
{% raw %}
import * as React from "react";
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Large} buttonSettings={{ iconCss: "e-icons e-
cut", content: "Cut" }}>
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
      </RibbonComponent>
    );
  }
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}
```

INDEX.TSX

```
{% raw %}
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective,
RibbonCollectionsDirective, RibbonCollectionDirective, RibbonGroupsDirective,
RibbonGroupDirective, RibbonItemsDirective, RibbonItemDirective,
RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
function App() {
    return (
        <RibbonComponent id="ribbon">
            <RibbonTabsDirective>
                <RibbonTabDirective header="Home">
                    <RibbonGroupsDirective>
                        <RibbonGroupDirective header="Clipboard">
                            <RibbonCollectionsDirective>
                                <RibbonCollectionDirective>
                                    <RibbonItemsDirective>
                                        <RibbonItemDirective type="Button"
allowedSizes={RibbonItemSize.Large} buttonSettings={{ iconCss: "e-icons e-
cut", content: "Cut" }}>
                                            </RibbonItemDirective>
                                        </RibbonItemsDirective>
                                    </RibbonCollectionDirective>
                                </RibbonCollectionsDirective>
                            </RibbonGroupDirective>
                        </RibbonGroupsDirective>
                    </RibbonTabDirective>
                </RibbonTabsDirective>
            </RibbonComponent>
        );
    }
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
{% endraw %}
```

INDEX.CSS

```
/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
```

Defining items active size

You can use the [activeSize](#) property to define the item size initially, before it is being resized. When resized the [activeSize](#) property is updated based on the ribbon's overflow state, which is determined by the [allowedSizes](#) property being configured. By default, the value is [Medium](#).

Events

This section describes the ribbon events that will be triggered when appropriate actions are performed. The following events are available in the ribbon component.

[tabSelected](#)

The [tabSelected](#) event is triggered after selecting the tab item.

```
{% raw %}
`ts
import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, TabSelectedEventArgs } from "@syncfusion/ej2-react-ribbon";

function App() {
function tabSelected ( args: TabSelectedEventArgs) {
// Your required actions here
}
return (
<RibbonComponent id="ribbon" tabSelected = { tabSelected }>
<RibbonTabsDirective>
<RibbonTabDirective header="Home" >
<RibbonGroupsDirective>
<RibbonGroupDirective header="Clipboard">
<RibbonCollectionsDirective>
<RibbonCollectionDirective>
<RibbonItemsDirective>
<RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
```

```
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`
```

```
{% endraw %}
```

tabSelecting

The [tabSelecting](#) event is triggered before selecting the tab item.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, TabSelectingEventArgs } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
function tabSelectingEvent ( args: TabSelectingEventArgs) {
```

```
// Your required actions here
```

```
}
```

```
return (
```

```
<RibbonComponent id="ribbon" tabSelecting = { tabSelectingEvent }>
```

```
<RibbonTabsDirective>
```

```
<RibbonTabDirective header="Home" >
```

```
<RibbonGroupsDirective>
```

```
<RibbonGroupDirective header="Clipboard">
```

```
<RibbonCollectionsDirective>
```

```
<RibbonCollectionDirective>
```

```
<RibbonItemsDirective>
```

```
<RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
```

```
</RibbonItemDirective>
```

```
</RibbonItemsDirective>
```

```
</RibbonCollectionDirective>
```

```
</RibbonCollectionsDirective>
```

```
</RibbonGroupDirective>
```

```
</RibbonGroupsDirective>
```

```
</RibbonTabDirective>
```

```

</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

ribbonCollapsing

The [ribbonCollapsing](#) event is triggered before collapsing the ribbon.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, ExpandCollapseEventArgs } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
function ribbonCollapsing ( args: ExpandCollapseEventArgs) {
```

```
// Your required actions here
```

```
}
```

```
return (
```

```
<RibbonComponent id="ribbon" ribbonCollapsing = { ribbonCollapsing }>
```

```
<RibbonTabsDirective>
```

```
<RibbonTabDirective header="Home" >
```

```
<RibbonGroupsDirective>
```

```
<RibbonGroupDirective header="Clipboard">
```

```
<RibbonCollectionsDirective>
```

```
<RibbonCollectionDirective>
```

```
<RibbonItemsDirective>
```

```
<RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
```

```
</RibbonItemDirective>
```

```
</RibbonItemsDirective>
```

```
</RibbonCollectionDirective>
```

```

</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[ribbonExpanding](#)

The [ribbonExpanding](#) event is triggered before expanding the ribbon.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, ExpandCollapseEventArgs } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
function ribbonExpanding ( args: ExpandCollapseEventArgs) {
```

```
// Your required actions here
```

```
}
```

```
return (
```

```
<RibbonComponent id="ribbon" ribbonExpanding = { ribbonExpanding }>
```

```
<RibbonTabsDirective>
```

```
<RibbonTabDirective header="Home" >
```

```
<RibbonGroupsDirective>
```

```
<RibbonGroupDirective header="Clipboard">
```

```
<RibbonCollectionsDirective>
```

```
<RibbonCollectionDirective>
```

```
<RibbonItemsDirective>
```

```

<RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[launcherIconClick](#)

The [launcherIconClick](#) event is triggered when the launcher icon of the group is clicked.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, LauncherClickEventArgs } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
function launchClick ( args: LauncherClickEventArgs) {
```

```
// Your required actions here
```

```
}
```

```
return (
```

```
<RibbonComponent id="ribbon" launcherIconClick = { launchClick }>
```

```
<RibbonTabsDirective>
```

```
<RibbonTabDirective header="Home" >
```

```
<RibbonGroupsDirective>
```

```

<RibbonGroupDirective header="Clipboard" showLauncherIcon={true}>
  <RibbonCollectionsDirective>
    <RibbonCollectionDirective>
      <RibbonItemsDirective>
        <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
        </RibbonItemDirective>
      </RibbonItemsDirective>
    </RibbonCollectionDirective>
  </RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));

```

```
{% endraw %}
```

Button item events

clicked

The [clicked](#) event is triggered when the button is clicked.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```

<RibbonTabDirective header="Home" >
  <RibbonGroupsDirective>
    <RibbonGroupDirective header="Clipboard">
      <RibbonCollectionsDirective>
        <RibbonCollectionDirective>
          <RibbonItemsDirective>
            <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" , clicked:
            function () {
              // Your required actions here
            }}}>
          </RibbonItemDirective>
        </RibbonItemsDirective>
      </RibbonCollectionDirective>
    </RibbonCollectionsDirective>
  </RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

created

The [created](#) event is triggered when the button is created.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```

function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" ,
                    created: function () {
                      // Your required actions here
                    }}}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
  );
}

export default App;
ReactDOM.render(<App />, document.getElementById("element"));

```

```
{% endraw %}
```

Checkbox item events

change

The [change](#) event is triggered when the Checkbox state is changed.

```
{% raw %}
```

```
`ts
```



```

import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";

function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="CheckBox" checkBoxSettings={{ label: "Ruler", checked: false, change:
function () {
// Your required actions here
}}} >
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
);
}

export default App;

ReactDOM.render(<App />, document.getElementById("element"));
`

{% endraw %}

```

created

The [created](#) event is triggered once the Checkbox is created.

```
{% raw %}
`ts
import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";

function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="CheckBox" checkBoxSettings={{ label: "Ruler", checked: false, created:
function () {
// Your required actions here
}}}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
  );
}
```

```
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`
```

```
{% endraw %}
```

Colorpicker item events

change

The [change](#) event is triggered while changing the colors.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { RibbonColorPicker, Inject } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Font">
```

```
              <RibbonCollectionsDirective>
```

```
                <RibbonCollectionDirective>
```

```
                  <RibbonItemsDirective>
```

```
                    <RibbonItemDirective type="ColorPicker" colorPickerSettings={{ value: "#123456" },change: function
(args) {
```

```
                      // Your required actions here
```

```
                    }}}>
```

```
                  </RibbonItemDirective>
```

```
                </RibbonItemsDirective>
```

```
              </RibbonCollectionDirective>
```

```
            </RibbonCollectionsDirective>
```

```
          </RibbonGroupDirective>
```

```
        </RibbonGroupsDirective>
```

```

</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonColorPicker]} />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

created

The [created](#) event is triggered once the ColorPicker is created.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { RibbonColorPicker, Inject } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Font">
```

```
              <RibbonCollectionsDirective>
```

```
                <RibbonCollectionDirective>
```

```
                  <RibbonItemsDirective>
```

```
                    <RibbonItemDirective type="ColorPicker" colorPickerSettings={{ value: "#123456" ,created: function
(args) {
```

```
                      // Your required actions here
```

```
                    }}}>
```

```

</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonColorPicker]} />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));

```

```
{% endraw %}
```

[open](#)

The [open](#) event is triggered while opening the ColorPicker popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { RibbonColorPicker, Inject } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Font">
```

```
              <RibbonCollectionsDirective>
```

```

<RibbonCollectionDirective>
  <RibbonItemsDirective>
    <RibbonItemDirective type="ColorPicker" colorPickerSettings={{ value: "#123456" ,open: function (args)
    {
    // Your required actions here
    }}}>
  </RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonColorPicker]} />
</RibbonComponent>
);
}

export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[select](#)

The [select](#) event is triggered while selecting the color in picker/palette, when showButtons property is enabled.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { RibbonColorPicker, Inject } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  return (
```

```

<RibbonComponent id="ribbon">
  <RibbonTabsDirective>
    <RibbonTabDirective header="Home" >
      <RibbonGroupsDirective>
        <RibbonGroupDirective header="Font">
          <RibbonCollectionsDirective>
            <RibbonCollectionDirective>
              <RibbonItemsDirective>
                <RibbonItemDirective type="ColorPicker" colorPickerSettings={{ value: "#123456" ,select: function (args)
                {
                // Your required actions here
                }}}>
              </RibbonItemDirective>
            </RibbonItemsDirective>
          </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
      </RibbonGroupDirective>
    </RibbonGroupsDirective>
  </RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonColorPicker]} />
</RibbonComponent>
);
}

export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[beforeClose](#)

The [beforeClose](#) event is triggered before closing the ColorPicker popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
import { RibbonColorPicker, Inject } from "@syncfusion/ej2-react-ribbon";
function App() {
return (
<RibbonComponent id="ribbon">
<RibbonTabsDirective>
<RibbonTabDirective header="Home" >
<RibbonGroupsDirective>
<RibbonGroupDirective header="Font">
<RibbonCollectionsDirective>
<RibbonCollectionDirective>
<RibbonItemsDirective>
<RibbonItemDirective type="ColorPicker" colorPickerSettings={{ value: "#123456" },beforeClose: function
(args) {
// Your required actions here
}}}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonColorPicker]} />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`
{% endraw %}

```


beforeOpen

The [beforeOpen](#) event is triggered before opening the ColorPicker popup.

```
{% raw %}
`ts
import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
import { RibbonColorPicker, Inject } from "@syncfusion/ej2-react-ribbon";

function App() {
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Font">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="ColorPicker" colorPickerSettings={{ value: "#123456" },beforeOpen: function
(args) {
// Your required actions here
}}}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
    <Inject services={[RibbonColorPicker]} />
  </RibbonComponent>
```

```
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`
```

```
{% endraw %}
```

beforeTileRender

The [beforeTileRender](#) event is triggered while rendering each palette tile.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { RibbonColorPicker, Inject } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Font">
```

```
              <RibbonCollectionsDirective>
```

```
                <RibbonCollectionDirective>
```

```
                  <RibbonItemsDirective>
```

```
                    <RibbonItemDirective type="ColorPicker" colorPickerSettings={{ value: "#123456" },beforeTileRender:
function (args) {
```

```
                      // Your required actions here
```

```
                    }}}>
```

```
                </RibbonItemDirective>
```

```
              </RibbonItemsDirective>
```

```
            </RibbonCollectionDirective>
```

```
          </RibbonCollectionsDirective>
```

```

</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonColorPicker]} />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));

```

```
{% endraw %}
```

ComboBox item events

change

The [change](#) event is triggered when an item in a popup is selected or when the model value is changed by the user.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
"Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
Roman", "Verdana", "Windings"];
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Font">
```

```
              <RibbonCollectionsDirective>
```

```
                <RibbonCollectionDirective>
```

```

<RibbonItemsDirective>
  <RibbonItemDirective type="ComboBox" comboBoxSettings={{ dataSource: fontStyle, index: 3, change:
function (args) {
  // Your required action here
} }}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[close](#)

The [close](#) event is triggered when the popup is closed.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
"Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
Roman", "Verdana", "Windings"];
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```

<RibbonTabsDirective>
<RibbonTabDirective header="Home" >
<RibbonGroupsDirective>
<RibbonGroupDirective header="Font">
<RibbonCollectionsDirective>
<RibbonCollectionDirective>
<RibbonItemsDirective>
<RibbonItemDirective type="ComboBox" comboBoxSettings={{ dataSource: fontStyle, index: 3, close:
function (args) {
// Your required action here
} }}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}

export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[open](#)

The [open](#) event is triggered when the popup is opened.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";

function App() {
  const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
"Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
Roman", "Verdana", "Windings"];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Font">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="ComboBox" comboBoxSettings={{ dataSource: fontStyle, index: 3, open:
function (args) {
// Your required action here
} }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
  );
}

export default App;

ReactDOM.render(<App />, document.getElementById("element"));
`

{% endraw %}

```

created

The [created](#) event is triggered once the Combobox is created.

```
{% raw %}
`ts
import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";

function App() {

const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
"Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
Roman", "Verdana", "Windings"];

return (

<RibbonComponent id="ribbon">
<RibbonTabsDirective>
<RibbonTabDirective header="Home" >
<RibbonGroupsDirective>
<RibbonGroupDirective header="Font">
<RibbonCollectionsDirective>
<RibbonCollectionDirective>
<RibbonItemsDirective>
<RibbonItemDirective type="ComboBox" comboBoxSettings={{ dataSource: fontStyle, index: 3, created:
function (args) {
// Your required action here
} }}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
```

```
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`
```

```
{% endraw %}
```

[filtering](#)

The [filtering](#) event triggers on typing a character in the Combobox.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
"Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
Roman", "Verdana", "Windings"];
```

```
return (
```

```
<RibbonComponent id="ribbon">
```

```
<RibbonTabsDirective>
```

```
<RibbonTabDirective header="Home" >
```

```
<RibbonGroupsDirective>
```

```
<RibbonGroupDirective header="Font">
```

```
<RibbonCollectionsDirective>
```

```
<RibbonCollectionDirective>
```

```
<RibbonItemsDirective>
```

```
<RibbonItemDirective type="ComboBox" comboBoxSettings={{ dataSource: fontStyle, index: 3, filtering:
function (args) {
```

```
// Your required action here
```

```
}}}>
```

```
</RibbonItemDirective>
```

```
</RibbonItemsDirective>
```

```
</RibbonCollectionDirective>
```



```

</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

select

The [select](#) event is triggered when an item in the popup is selected.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
  "Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
  Roman", "Verdana", "Windings"];

```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Font">
```

```
              <RibbonCollectionsDirective>
```

```
                <RibbonCollectionDirective>
```

```
                  <RibbonItemsDirective>
```

```

<RibbonItemDirective type="ComboBox" comboBoxSettings={{ dataSource: fontStyle, index: 3, select:
function (args) {
// Your required action here
} }}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[beforeOpen](#)

The [beforeOpen](#) event triggers before opening the popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
const fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
"Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
Roman", "Verdana", "Windings"];
```

```
return (
```

```
<RibbonComponent id="ribbon">
```

```
<RibbonTabsDirective>
```

```

<RibbonTabDirective header="Home" >
  <RibbonGroupsDirective>
    <RibbonGroupDirective header="Font">
      <RibbonCollectionsDirective>
        <RibbonCollectionDirective>
          <RibbonItemsDirective>
            <RibbonItemDirective type="ComboBox" comboBoxSettings={{ dataSource: fontStyle, index: 3,
            beforeOpen: function (args) {
              // Your required action here
            }}}>
          </RibbonItemDirective>
        </RibbonItemsDirective>
      </RibbonCollectionDirective>
    </RibbonCollectionsDirective>
  </RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));

```

```
{% endraw %}
```

DropDown item events

beforeClose

The [beforeClose](#) event is triggered before closing the DropdownButton popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```

import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
  const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Tables">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="DropDown" dropDownSettings={{ iconCss: "e-icons e-table", items: tableOptions, content: "Table", beforeClose: function (args) {
                      // Your required action here
                    }}}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`
{% endraw %}

```

beforeOpen

The [beforeOpen](#) event is triggered before opening the Dropdown button popup.

```

{% raw %}
`ts
import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";

function App() {
  const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table"
}, { text: "Excel SpreadSheet" }];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Tables">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="DropDown" dropDownSettings={{ iconCss: "e-icons e-table", items:
tableOptions, content: "Table", beforeOpen: function (args) {
// Your required action here
} }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
);

```

```

}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

beforeItemRender

The [beforeItemRender](#) event is triggered while rendering each popup item of the Dropdown button.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
```

```
function App() {
```

```
  const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table"
}, { text: "Excel SpreadSheet" }];
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Tables">
```

```
              <RibbonCollectionsDirective>
```

```
                <RibbonCollectionDirective>
```

```
                  <RibbonItemsDirective>
```

```
                    <RibbonItemDirective type="DropDown" dropDownSettings={{ iconCss: "e-icons e-table", items:
tableOptions, content: "Table", beforeItemRender: function (args) {
```

```
                      // Your required action here
```

```
                    } }>
```

```
                  </RibbonItemDirective>
```

```
                </RibbonItemsDirective>
```

```
              </RibbonCollectionDirective>
```

```
            </RibbonCollectionsDirective>
```

```

</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

open

The [open](#) event is triggered while opening the Dropdown button popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
```

```
function App() {
```

```
  const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table"
}, { text: "Excel SpreadSheet" }];
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Tables">
```

```
              <RibbonCollectionsDirective>
```

```
                <RibbonCollectionDirective>
```

```
                  <RibbonItemsDirective>
```

```
                    <RibbonItemDirective type="DropDown" dropDownSettings={{ iconCss: "e-icons e-table", items:
tableOptions, content: "Table", open: function (args) {
```

```
// Your required action here
}}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`
```

```
{% endraw %}
```

[close](#)

The [close](#) event is triggered while closing the Dropdown button popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
```

```
function App() {
```

```
const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table"
}, { text: "Excel SpreadSheet" }];
```

```
return (
```

```
<RibbonComponent id="ribbon">
```

```
<RibbonTabsDirective>
```

```
<RibbonTabDirective header="Home" >
```



```

<RibbonGroupsDirective>
<RibbonGroupDirective header="Tables">
<RibbonCollectionsDirective>
<RibbonCollectionDirective>
<RibbonItemsDirective>
<RibbonItemDirective type="DropDown" dropDownSettings={{ iconCss: "e-icons e-table", items:
tableOptions, content: "Table", close: function (args) {
// Your required action here
} }}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));

```

```
{% endraw %}
```

created

The [created](#) event is triggered when the DropDown is created.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
```

```

function App() {
  const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table"
}, { text: "Excel SpreadSheet" }];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Tables">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="DropDown" dropDownSettings={{ iconCss: "e-icons e-table", items:
tableOptions, content: "Table", created: function (args) {
// Your required action here
} }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

{% endraw %}
select
The select event is triggered while selecting an action item in the Dropdown button popup.
{% raw %}

```

```

`ts
import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";

import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";

function App() {
  const tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table"
}, { text: "Excel SpreadSheet" }];

  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Tables">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="DropDown" dropDownSettings={{ iconCss: "e-icons e-table", items:
tableOptions, content: "Table", select: function (args) {
// Your required action here
}}} >
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
  );
}

```

```
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`
```

```
{% endraw %}
```

SplitButton item events

beforeClose

The [beforeClose](#) event is triggered before closing the SplitButton popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
```

```
function App() {
```

```
  const selectOptions: ItemModel[] = [{ text: "Select All" }, { text: "Select Objects" }];
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Clipboard">
```

```
              <RibbonCollectionsDirective>
```

```
                <RibbonCollectionDirective>
```

```
                  <RibbonItemsDirective>
```

```
                    <RibbonItemDirective type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
items: selectOptions, content: "Select", beforeClose: function (args) {
```

```
                      // Your required action here
```

```
                    }}}>
```

```
                </RibbonItemDirective>
```

```
              </RibbonItemsDirective>
```

```
            </RibbonCollectionDirective>
```

```
          </RibbonCollectionsDirective>
```

```
        </RibbonGroupDirective>
```

```

</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

beforeOpen

The [beforeOpen](#) event is triggered before opening the SplitButton popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
```

```
function App() {
```

```
  const selectOptions: ItemModel[] = [{ text: "Select All" }, { text: "Select Objects" }];
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Clipboard">
```

```
              <RibbonCollectionsDirective>
```

```
                <RibbonCollectionDirective>
```

```
                  <RibbonItemsDirective>
```

```
                    <RibbonItemDirective type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
items: selectOptions, content: "Select", beforeOpen: function (args) {
```

```
// Your required action here
```

```

    >>>
  </RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

beforeItemRender

The [beforeItemRender](#) event is triggered while rendering each popup item of SplitButton.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
```

```
function App() {
```

```
  const selectOptions: ItemModel[] = [{ text: "Select All" }, { text: "Select Objects" }];
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Clipboard">
```

```

<RibbonCollectionsDirective>
<RibbonCollectionDirective>
<RibbonItemsDirective>
<RibbonItemDirective type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
items: selectOptions, content: "Select", beforeItemRender: function (args) {
// Your required action here
}}}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[open](#)

The [open](#) event is triggered while opening the SplitButton popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
```

```
function App() {
```

```
const selectOptions: ItemModel[] = [{ text: "Select All" }, { text: "Select Objects" }];
```

```

return (
  <RibbonComponent id="ribbon">
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home" >
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Clipboard">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
                    items: selectOptions, content: "Select", open: function (args) {
                      // Your required action here
                    }}}>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>
</RibbonComponent>
);
}

export default App;
ReactDOM.render(<App />, document.getElementById("element"));

```

```
{% endraw %}
```

[close](#)

The [close](#) event is triggered while closing the SplitButton popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```



```

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
function App() {
const selectOptions: ItemModel[] = [{ text: "Select All" }, { text: "Select Objects" }];
return (
<RibbonComponent id="ribbon">
<RibbonTabsDirective>
<RibbonTabDirective header="Home" >
<RibbonGroupsDirective>
<RibbonGroupDirective header="Clipboard">
<RibbonCollectionsDirective>
<RibbonCollectionDirective>
<RibbonItemsDirective>
<RibbonItemDirective type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
items: selectOptions, content: "Select", close: function (args) {
// Your required action here
}}}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`
{% endraw %}

```

created

The [created](#) event is triggered when the SplitButton is created.

```
{% raw %}
`ts
import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";

function App() {
  const selectOptions: ItemModel[] = [{ text: "Select All" }, { text: "Select Objects" }];
  return (
    <RibbonComponent id="ribbon">
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home" >
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
items: selectOptions, content: "Select", created: function (args) {
// Your required action here
}}}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
  </RibbonComponent>
```

```
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`
```

```
{% endraw %}
```

[select](#)

The [select](#) event is triggered while selecting an action item in the SplitButton popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
```

```
function App() {
```

```
  const selectOptions: ItemModel[] = [{ text: "Select All" }, { text: "Select Objects" }];
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Clipboard">
```

```
              <RibbonCollectionsDirective>
```

```
                <RibbonCollectionDirective>
```

```
                  <RibbonItemsDirective>
```

```
                    <RibbonItemDirective type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
items: selectOptions, content: "Select", select: function (args) {
```

```
                    // Your required action here
```

```
                    }}}>
```

```
                </RibbonItemDirective>
```

```
              </RibbonItemsDirective>
```

```
            </RibbonCollectionDirective>
```

```

</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[click](#)

The [click](#) event is triggered while clicking the primary button in the SplitButton.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective } from "@syncfusion/ej2-react-ribbon";
```

```
import { ItemModel } from "@syncfusion/ej2-react-splitbuttons";
```

```
function App() {
```

```
  const selectOptions: ItemModel[] = [{ text: "Select All" }, { text: "Select Objects" }];
```

```
  return (
```

```
    <RibbonComponent id="ribbon">
```

```
      <RibbonTabsDirective>
```

```
        <RibbonTabDirective header="Home" >
```

```
          <RibbonGroupsDirective>
```

```
            <RibbonGroupDirective header="Clipboard">
```

```
              <RibbonCollectionsDirective>
```

```
                <RibbonCollectionDirective>
```

```
                  <RibbonItemsDirective>
```

```

<RibbonItemDirective type="SplitButton" splitButtonSettings={{ iconCss: "e-icons e-mouse-pointer",
items: selectOptions, content: "Select", click: function (args) {
// Your required action here
}}}>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

GroupButton item events

[beforeClick](#)

The [beforeClick](#) event is triggered before selecting a button from the groupbutton items.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, RibbonGroupButtonSelection , RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  const groupButtonMultiple: RibbonGroupButtonSettingsModel = {
```

```
    selection: RibbonGroupButtonSelection.Multiple,
```

```
    items: [
```

```
      { iconCss: 'e-icons e-bold', content: 'Bold', selected: true, beforeClick: () => {
```

```

// Your required action here
alert("")
}},
{ iconCss: 'e-icons e-italic', content: 'Italic', beforeClick: () => {
// Your required action here
}},
{ iconCss: 'e-icons e-underline', content: 'Underline', beforeClick: () => {
// Your required action here
}},
{ iconCss: 'e-icons e-strikethrough', content: 'Strikethrough', beforeClick: () => {
// Your required action here
}}, { iconCss: 'e-icons e-change-case', content: 'Change Case', beforeClick: () => {
// Your required action here
}}
]
}
return (
<RibbonComponent id="ribbon">
<RibbonTabsDirective>
<RibbonTabDirective header="Home" >
<RibbonGroupsDirective>
<RibbonGroupDirective header="Paragraph">
<RibbonCollectionsDirective>
<RibbonCollectionDirective>
<RibbonItemsDirective>
<RibbonItemDirective type="GroupButton" allowedSizes={RibbonItemSize.Medium}
groupButtonSettings={ groupButtonMultiple }>
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>

```

```

</RibbonTabDirective>
</RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[click](#)

The [click](#) event is triggered when selecting a button from the groupbutton items.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, RibbonGroupButtonSelection , RibbonItemSize } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  const groupButtonSingle: RibbonGroupButtonSettingsModel = {
```

```
    selection: RibbonGroupButtonSelection.Single,
```

```
    items: [
```

```
      {iconCss: 'e-icons e-align-left', content: 'Align Left', selected: true, click: () => {
```

```
        // Your required action here
```

```
      }},
```

```
      {iconCss: 'e-icons e-align-center',content: 'Align Center', click: () => {
```

```
        // Your required action here
```

```
      }},
```

```
      {iconCss: 'e-icons e-align-right',content: 'Align Right', click: () => {
```

```
        // Your required action here
```

```
      }},
```

```
      {iconCss: 'e-icons e-justify',content: 'Justify', click: () => {
```

```
        // Your required action here
```

```

  }}
  ]
}
return (
  <RibbonComponent id="ribbon">
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home" >
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Paragraph">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="GroupButton" allowedSizes={RibbonItemSize.Medium}
                    groupButtonSettings={ groupButtonSingle }>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

FileMenu events

beforeClose

The [beforeClose](#) event is triggered before closing the FileMenu popup.

```
{% raw %}
```

```
`ts
```



```
import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, FileMenuSettingsModel } from "@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
import { MenuItemModel } from '@syncfusion/ej2-react-navigations';

function App() {
  const fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  { text: "Save as", iconCss: "e-icons e-save", id: "save" }]
  const filesettings : FileMenuSettingsModel = {
    menuItems: fileOptions,
    visible: true,
    beforeClose: () => {
      // Your required action here
    }
  };
  return (
    <RibbonComponent id="ribbon" fileMenu= {filesettings}>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
    </RibbonComponent>
  );
}
```

```

</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonFileMenu]} />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

beforeOpen

The [beforeOpen](#) event is triggered before opening the FileMenu popup.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, FileMenuSettingsModel } from "@syncfusion/ej2-react-ribbon";
```

```
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
```

```
import { MenuItemModel } from '@syncfusion/ej2-react-navigations';
```

```
function App() {
```

```
  const fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
```

```
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
```

```
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
```

```
  { text: "Save as", iconCss: "e-icons e-save", id: "save" } ]
```

```
  const fileSettings : FileMenuSettingsModel = {
```

```
    menuItems: fileOptions,
```

```
    visible: true,
```

```
    beforeOpen: () => {
```

```
      // Your required action here
```

```
    }
```

```
  };
```

```

return (
  <RibbonComponent id="ribbon" fileMenu= {filesettings}>
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home">
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Clipboard">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>
  <Inject services={[RibbonFileMenu]} />
</RibbonComponent>
);
}

export default App;
ReactDOM.render(<App />, document.getElementById("element"));

```

```
{% endraw %}
```

beforeItemRender

The [beforeItemRender](#) event is triggered while rendering each ribbon FileMenu item.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, FileMenuSettingsModel } from "@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
import { MenuItemModel } from '@syncfusion/ej2-react-navigations';

function App() {
  const fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  { text: "Save as", iconCss: "e-icons e-save", id: "save" }]
  const filesettings : FileMenuSettingsModel = {
    menuItems: fileOptions,
    visible: true,
    beforeItemRender: () => {
      // Your required action here
    }
  };
  return (
    <RibbonComponent id="ribbon" fileMenu= {filesettings}>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                  </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonComponent>
  );
}

```

```

</RibbonTabsDirective>
<Inject services={[RibbonFileMenu]} />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[open](#)

The [open](#) event is triggered when the FileMenu popup is opened.

```
{% raw %}
```

```
`ts
```

```

import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, FileMenuSettingsModel } from "@syncfusion/ej2-react-ribbon";
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
import { MenuItemModel } from '@syncfusion/ej2-react-navigations';

function App() {
  const fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  { text: "Save as", iconCss: "e-icons e-save", id: "save" }]
  const filesettings : FileMenuSettingsModel = {
    menuItems: fileOptions,
    visible: true,
    open: () => {
      // Your required action here
    }
  };
  return (
    <RibbonComponent id="ribbon" fileMenu= {filesettings}>

```

```

<RibbonTabsDirective>
  <RibbonTabDirective header="Home">
    <RibbonGroupsDirective>
      <RibbonGroupDirective header="Clipboard">
        <RibbonCollectionsDirective>
          <RibbonCollectionDirective>
            <RibbonItemsDirective>
              <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
            </RibbonItemDirective>
          </RibbonItemsDirective>
        </RibbonCollectionDirective>
      </RibbonCollectionsDirective>
    </RibbonGroupDirective>
  </RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonFileMenu]} />
</RibbonComponent>
);
}

export default App;
ReactDOM.render(<App />, document.getElementById("element"));

```

```
{% endraw %}
```

[close](#)

The [close](#) event is triggered when the FileMenu popup is closed.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, FileMenuSettingsModel } from "@syncfusion/ej2-react-ribbon";
```

```
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
```

```

import { MenuItemModel } from '@syncfusion/ej2-react-navigations';
function App() {
  const fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  { text: "Save as", iconCss: "e-icons e-save", id: "save" }]
  const fileSettings : FileMenuSettingsModel = {
    menuItems: fileOptions,
    visible: true,
    close: () => {
      // Your required action here
    }
  };
  return (
    <RibbonComponent id="ribbon" fileMenu= {fileSettings}>
      <RibbonTabsDirective>
        <RibbonTabDirective header="Home">
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Clipboard">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                    </RibbonItemDirective>
                  </RibbonItemsDirective>
                </RibbonCollectionDirective>
              </RibbonCollectionsDirective>
            </RibbonGroupDirective>
          </RibbonGroupsDirective>
        </RibbonTabDirective>
      </RibbonTabsDirective>
      <Inject services={[RibbonFileMenu]} />
    </RibbonComponent>
  )
}

```

```
);
}

export default App;

ReactDOM.render(<App />, document.getElementById("element"));
`
```

```
{% endraw %}
```

[select](#)

The [select](#) event is triggered while selecting an item in the ribbon FileMenu.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, FileMenuSettingsModel } from "@syncfusion/ej2-react-ribbon";
```

```
import { RibbonFileMenu, Inject } from "@syncfusion/ej2-react-ribbon";
```

```
import { MenuItemModel } from '@syncfusion/ej2-react-navigations';
```

```
function App() {
```

```
  const fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
```

```
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
```

```
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
```

```
  { text: "Save as", iconCss: "e-icons e-save", id: "save" } ]
```

```
  const filesettings : FileMenuSettingsModel = {
```

```
    menuItems: fileOptions,
```

```
    visible: true,
```

```
    select: () => {
```

```
      // Your required action here
```

```
    }
```

```
  };
```

```
  return (
```

```
    <RibbonComponent id="ribbon" fileMenu= {filesettings}>
```

```
    <RibbonTabsDirective>
```

```
    <RibbonTabDirective header="Home">
```

```
    <RibbonGroupsDirective>
```



```

<RibbonGroupDirective header="Clipboard">
  <RibbonCollectionsDirective>
    <RibbonCollectionDirective>
      <RibbonItemsDirective>
        <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
        </RibbonItemDirective>
      </RibbonItemsDirective>
    </RibbonCollectionDirective>
  </RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={{[RibbonFileMenu]} } />
</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

Backstage view events

[backStageItemClick](#)

The [backStageItemClick](#) event is triggered when backstage item is selected.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, RibbonBackstage , Inject, BackStageMenuModel } from "@syncfusion/ej2-react-ribbon";
```

```
function App() {
```

```
  const backstageSettings: BackStageMenuModel = {
```

```
    visible: true,
```

```

items: [
  { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content: homeContentTemplate(),
    backStageItemClick: () => {
      // Your required action here
    } },
],
backButton: {
  text: 'Close',
}
}
return (
  <RibbonComponent id="backstage-ribbon" backstageMenu={backstageSettings}>
    <RibbonTabsDirective>
      <RibbonTabDirective header="Home" >
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Paragraph">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-cut", content: "Cut" }}>
                </RibbonItemDirective>
                  <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-copy", content: "Copy" }}>
                </RibbonItemDirective>
                  <RibbonItemDirective type="Button" buttonSettings={{ iconCss: "e-icons e-paste", content: "Paste" }}>
                </RibbonItemDirective>
                </RibbonItemsDirective>
              </RibbonCollectionDirective>
            </RibbonCollectionsDirective>
          </RibbonGroupDirective>
        </RibbonGroupsDirective>
      </RibbonTabDirective>
    </RibbonTabsDirective>
    <Inject services={[RibbonBackstage]} />
  </RibbonComponent>
)

```

```

</RibbonComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));

```

```
{% endraw %}
```

Gallery events

popupOpen

The [popupOpen](#) event is triggered when the gallery popup opens.

```
{% raw %}
```

```
`ts
```

```

import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, Inject, RibbonGallery, RibbonGallerySettingsModel, GalleryPopupEventArgs } from
'@syncfusion/ej2-react-ribbon';

function App() {
  const gallerySettings: RibbonGallerySettingsModel = (
    {
      groups: [{
        header: 'Styles',
        items: [
          {
            content: 'Normal'
          }, {
            content: 'No Spacing'
          }, {
            content: 'Heading 1'
          }, {
            content: 'Heading 2'
          }
        ]
      }
    ]
  ),

```

```

popupOpen: (args: GalleryPopupEventArgs) => {
  // Your required action here
}
}
);
return (
<div>
  <RibbonComponent id='ribbon'>
    <RibbonTabsDirective>
      <RibbonTabDirective header='Home'>
        <RibbonGroupsDirective>
          <RibbonGroupDirective header="Gallery">
            <RibbonCollectionsDirective>
              <RibbonCollectionDirective>
                <RibbonItemsDirective>
                  <RibbonItemDirective type="Gallery" gallerySettings={gallerySettings} >
                </RibbonItemDirective>
              </RibbonItemsDirective>
            </RibbonCollectionDirective>
          </RibbonCollectionsDirective>
        </RibbonGroupDirective>
      </RibbonGroupsDirective>
    </RibbonTabDirective>
  </RibbonTabsDirective>
  <Inject services={[RibbonGallery]} />
</RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
,

{% enddraw %}

```

popupClose

The [popupClose](#) event is triggered when the gallery popup closes.

```
{% raw %}
`ts
import * as React from "react";
import * as ReactDOM from "react-dom";

import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, Inject, RibbonGallery, RibbonGallerySettingsModel, GalleryPopupEventArgs } from
'@syncfusion/ej2-react-ribbon';

function App() {
  const gallerySettings: RibbonGallerySettingsModel = (
    {
      groups: [{
        header: 'Styles',
        items: [
          {
            content: 'Normal'
          }, {
            content: 'No Spacing'
          }, {
            content: 'Heading 1'
          }, {
            content: 'Heading 2'
          }
        ]
      }],
      popupClose: (args: GalleryPopupEventArgs) => {
        // Your required action here
      }
    }
  );
  return (
    <div>
```

```

<RibbonComponent id='ribbon'>
  <RibbonTabsDirective>
    <RibbonTabDirective header='Home'>
      <RibbonGroupsDirective>
        <RibbonGroupDirective header="Gallery">
          <RibbonCollectionsDirective>
            <RibbonCollectionDirective>
              <RibbonItemsDirective>
                <RibbonItemDirective type="Gallery" gallerySettings={gallerySettings} >
              </RibbonItemDirective>
            </RibbonItemsDirective>
          </RibbonCollectionDirective>
        </RibbonCollectionsDirective>
      </RibbonGroupDirective>
    </RibbonGroupsDirective>
  </RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonGallery]} />
</RibbonComponent>
</div>
);
}

export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

itemHover

The [itemHover](#) event is triggered when hover over the gallery item.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
```

```
RibbonItemDirective, Inject, RibbonGallery, RibbonGallerySettingsModel, GalleryHoverEventArgs } from '@syncfusion/ej2-react-ribbon';
```

```
function App() {
  const gallerySettings: RibbonGallerySettingsModel = (
    {
      groups: [{
        header: 'Styles',
        items: [
          {
            content: 'Normal'
          }, {
            content: 'No Spacing'
          }, {
            content: 'Heading 1'
          }, {
            content: 'Heading 2'
          }
        ]
      }],
      itemHover: (args: GalleryHoverEventArgs) => {
        // Your required action here
      }
    );
  return (
    <div>
      <RibbonComponent id='ribbon'>
        <RibbonTabsDirective>
          <RibbonTabDirective header='Home'>
            <RibbonGroupsDirective>
              <RibbonGroupDirective header="Gallery">
                <RibbonCollectionsDirective>
                  <RibbonCollectionDirective>
```

```

<RibbonItemsDirective>
<RibbonItemDirective type="Gallery" gallerySettings={gallerySettings} >
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonGallery]} />
</RibbonComponent>
</div>
);
}

export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

[*beforeItemRender*](#)

The [beforeItemRender](#) event is triggered while rendering each gallery item.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, Inject, RibbonGallery, RibbonGallerySettingsModel, GalleryItemEventArgs } from
'@syncfusion/ej2-react-ribbon';
```

```
function App() {
```

```
  const gallerySettings: RibbonGallerySettingsModel = (
```

```
  {
```

```
    groups: [{
```

```
      header: 'Styles',
```



```

items: [
  {
    content: 'Normal'
  }, {
    content: 'No Spacing'
  }, {
    content: 'Heading 1'
  }, {
    content: 'Heading 2'
  }
]
}},
beforeItemRender: (args: GalleryItemEventArgs) => {
  // Your required action here
}
}
);
return (
  <div>
    <RibbonComponent id='ribbon'>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Gallery">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Gallery" gallerySettings={gallerySettings} >
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
      </RibbonComponent>
    </div>
  );
}
}

```

```

</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonGallery]} />
</RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

beforeSelect

The [beforeSelect](#) event is triggered before selecting an item in the Ribbon gallery.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, Inject, RibbonGallery, RibbonGallerySettingsModel, GalleryBeforeSelectEventArgs
} from '@syncfusion/ej2-react-ribbon';
```

```
function App() {
```

```
  const gallerySettings: RibbonGallerySettingsModel = (
```

```
  {
```

```
    groups: [{
```

```
      header: 'Styles',
```

```
      items: [
```

```
        {
```

```
          content: 'Normal'
```

```
        }, {
```

```
          content: 'No Spacing'
```

```
        }, {
```

```
          content: 'Heading 1'
```

```
}, {
  content: 'Heading 2'
}
],
}},
beforeSelect: (args: GalleryBeforeSelectEventArgs) => {
  // Your required action here
}
}
);
return (
  <div>
    <RibbonComponent id='ribbon'>
      <RibbonTabsDirective>
        <RibbonTabDirective header='Home'>
          <RibbonGroupsDirective>
            <RibbonGroupDirective header="Gallery">
              <RibbonCollectionsDirective>
                <RibbonCollectionDirective>
                  <RibbonItemsDirective>
                    <RibbonItemDirective type="Gallery" gallerySettings={gallerySettings} >
                      </RibbonItemDirective>
                    </RibbonItemsDirective>
                  </RibbonCollectionDirective>
                </RibbonCollectionsDirective>
              </RibbonGroupDirective>
            </RibbonGroupsDirective>
          </RibbonTabDirective>
        </RibbonTabsDirective>
        <Inject services={[RibbonGallery]} />
      </RibbonComponent>
    </div>
  );
```

```

}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`

```

```
{% endraw %}
```

select

The [select](#) event is triggered while selecting an item in the Ribbon Gallery.

```
{% raw %}
```

```
`ts
```

```
import * as React from "react";
```

```
import * as ReactDOM from "react-dom";
```

```
import { RibbonComponent, RibbonTabsDirective, RibbonTabDirective, RibbonCollectionsDirective,
RibbonCollectionDirective, RibbonGroupsDirective, RibbonGroupDirective, RibbonItemsDirective,
RibbonItemDirective, Inject, RibbonGallery, RibbonGallerySettingsModel, GallerySelectEventArgs } from
'@syncfusion/ej2-react-ribbon';
```

```
function App() {
```

```
const gallerySettings: RibbonGallerySettingsModel = (
```

```
{
```

```
groups: [{
```

```
header: 'Styles',
```

```
items: [
```

```
{
```

```
content: 'Normal'
```

```
}, {
```

```
content: 'No Spacing'
```

```
}, {
```

```
content: 'Heading 1'
```

```
}, {
```

```
content: 'Heading 2'
```

```
}
```

```
]
```

```
}],
```

```
select: (args: GallerySelectEventArgs) => {
```

```
// Your required action here
```

```

}
}
);
return (
<div>
<RibbonComponent id='ribbon'>
<RibbonTabsDirective>
<RibbonTabDirective header='Home'>
<RibbonGroupsDirective>
<RibbonGroupDirective header="Gallery">
<RibbonCollectionsDirective>
<RibbonCollectionDirective>
<RibbonItemsDirective>
<RibbonItemDirective type="Gallery" gallerySettings={gallerySettings} >
</RibbonItemDirective>
</RibbonItemsDirective>
</RibbonCollectionDirective>
</RibbonCollectionsDirective>
</RibbonGroupDirective>
</RibbonGroupsDirective>
</RibbonTabDirective>
</RibbonTabsDirective>
<Inject services={[RibbonGallery]} />
</RibbonComponent>
</div>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById("element"));
`
{% endraw %}

```

Accessibility in React Ribbon component

The Ribbon component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Ribbon component is outlined below.

| [Accessibility Criteria](#) | [Compatibility](#) |

| -- | -- |

| [WCAG 2.2](#) Support | `<img`

`src="https://cdn.syncfusion.com/content/images/documentation/partial.png" alt="Intermediate"> |`

| [Section 508](#) Support | `<img`

`src="https://cdn.syncfusion.com/content/images/documentation/full.png" alt="Yes"> |`

| Screen Reader Support | `<img`

`src="https://cdn.syncfusion.com/content/images/documentation/partial.png" alt="Intermediate"> |`

| Right-To-Left Support | `<img`

`src="https://cdn.syncfusion.com/content/images/documentation/full.png" alt="Yes"> |`

| Color Contrast | ` |`

| Mobile Device Support | `<img`

`src="https://cdn.syncfusion.com/content/images/documentation/full.png" alt="Yes"> |`

| Keyboard Navigation Support | `<img`

`src="https://cdn.syncfusion.com/content/images/documentation/full.png" alt="Yes"> |`

| [Accessibility Checker](#) Validation | `<img`

`src="https://cdn.syncfusion.com/content/images/documentation/full.png" alt="Yes"> |`

| [Axe-core](#) Accessibility Validation | `<img`

`src="https://cdn.syncfusion.com/content/images/documentation/full.png" alt="Yes"> |`

`<style>`

`.post .post-content img {`

`display: inline-block;`

`margin: 0.5em 0;`

`}`

`</style>`

`<div> - All features of the component meet the requirement.</div>`

`<div> - Some features of the component do not meet the requirement.</div>`

`<div> - The component does not meet the requirement.</div>`

[WAI-ARIA attributes](#)

The Ribbon component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Ribbon component:

| [Attributes](#) | [Purpose](#) |

| --- | --- |

| **role=tablist** | Used to identify the element that serves as the container for a set of tabs. |

| **role=tab** | Indicates an interactive element within a tablist that, when activated, displays its associated tab panel. |

| **role=tabpanel** | Specifies the role for the content associated with an active tab, describing its role in presenting the active content. |

| **role=button** | Represents a clickable element that trigger a response when activated by the user. |

| **role=menu** | Represents an item that have sub menu. |

| **role=menuitem** | Indicates an option in a set of choices within a menu. |

| **role=combobox** | Identifies an element as an input that controls another element, commonly used for dropdowns. |

| **role=option** | Used for selectable items in a combobox. |

| **role=gridcell** | Specified as gridcell for the tiles in the color palette. |

| **aria-orientation** | Indicates the element's orientation as horizontal, vertical, or unknown/ambiguous. |

| **aria-selected** | Indicates the current "selected" state of various widgets. |

| **aria-labelledby** | Sets to the Tab content element to indicates the associated Tab header for the content. |

| **aria-controls** | Indicates the associated tabpanel for the header by setting the attribute on Tab items. |

| **aria-haspopup** | Indicates availability and type of interactive popup triggered by the element it's set on. |

| **aria-disabled** | Indicates that the element is perceivable but disabled, making it not editable or operable. |

| **aria-expanded** | Indicates whether a component is expanded or collapsed, set on the respective element. |

| **aria-label** | Defines a string value that labels an interactive element for accessibility. |

| **aria-checked** | Indicates the current "checked" state of checkboxes, radio buttons, and other widgets. |

| **aria-owns** | Identifies an element or elements, establishing a relationship when DOM hierarchy can't represent it. |

| **aria-readonly** | Indicates that the element is not editable but is otherwise operable. |

| **aria-activedescendent** | Identifies the currently active element when focus is on a combobox, textbox, group, or application. |

| **aria-autocomplete** | Indicates whether inputting text could trigger display of predictions and specifies how predictions will be presented for a combobox, searchbox, or textbox. |

Keyboard interaction

The Ribbon component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Ribbon component.

| Press | To do this |

| --- | --- |

Ribbon Tab| |

| Tab | To focus the ribbon tabs. |

| Right Arrow | Moves focus to the next Tab. |

| Left Arrow | Moves focus to the previous Tab. |

| Enter / Space | To select the currently focussed ribbon tab. |

Ribbon Items| |

| Tab | To focus the ribbon Items. |

| Right Arrow | Focuses the next item. |

| Left Arrow | Focuses the previous item. |

| Enter / Space | To select the currently focussed ribbon item. |

Ribbon Dropdown Items/ Ribbon Split button| |

| Esc | Closes the popup. |

| Enter / Space | Opens the popup, or activates the highlighted item and closes the popup. |

| Arrow Up | Focuses the next item. |

| Arrow Down | Focuses the previous item. |

| Alt + Arrow Up | Closes the popup. |

| Alt + Arrow Down | Opens the popup |

Ribbon File menu| |

| Tab | To focus the ribbon file menu. |

| Esc | Closes the popup. |

| Enter | Opens the popup, or activates the highlighted item and closes the popup. |

| Arrow Up | Focuses the previous action item. |

| Arrow Down | Focuses the next action item. |

| Alt + Arrow Down | Opens the popup |

Ribbon Combobox| |

| Arrow Down | Selects the first item in the ComboBox when no item selected. Otherwise, selects the item next to the currently selected item. |

- | Arrow Up | Selects the item previous to the currently selected one. |
- | Page Down | Scrolls down to the next page and selects the first item when popup list opens. |
- | Page Up | Scrolls up to the previous page and selects the first item when popup list opens. |
- | Enter | Selects the focused item and popup list closes when it is in open state. |
- | Tab | Focuses on the next TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |
- | Shift + tab | Focuses on the previous TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |
- | Alt + Down | Open the popup list |
- | Alt + Up | Close the popup list |
- | Esc(Escape) | Closes the popup list when it is in an open state and the currently selected item remains the same. |
- | Home | Cursor moves to before of first character in input |
- | End | Cursor moves to next of last character in input |

Ensuring accessibility

The Ribbon component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Ribbon component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Ribbon component with accessibility tools.

See also

- [Accessibility in Syncfusion React components](#)

RichTextEditor

Getting Started with React Rich Text Editor Component

This article provides a step-by-step introduction to get started with the Syncfusion React Rich Text Editor component.

Prerequisites

[System requirements for Syncfusion React UI components](#)

Dependencies

The following list of dependencies are required to use the Rich Text Editor component in the application.

```
`javascript
|-- @syncfusion/ej2-react-richtexteditor
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
```

```
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-filemanager
|-- @syncfusion/ej2-richtexteditor
|-- @syncfusion/ej2-react-base
\
```

Create the React application

To set-up, a React application, choose any of the following ways. The best and easiest way is to use the [Create-react-app](#). It sets up the development environment in JavaScript and improvises the application for production. Refer to the [installation instructions](#) of the `Create-react-app`

```
`bash
npx create-react-app my-app
cd my-app
npm start
\
```

or

```
`bash
yarn create react-app my-app
cd my-app
yarn start
\
```

To set-up a React application in the `TypeScript` environment, run the following command.

```
`bash
npx create-react-app my-app --template typescript
cd my-app
npm start
\
```

Besides using the [npm](#) package runner tool, also create an application from the `npm init`. To begin with `npm init`, upgrade the `npm` version to `npm 6+`.

```
`bash
```

```
npm init react-app my-app
```

```
cd my-app
```

```
npm start
```

```
,
```

Add Syncfusion React packages

Once you have created the React application, install the required Syncfusion React component package in the application. All Syncfusion React (Essential JS 2) packages are published on the [npmjs](https://www.npmjs.com/) public registry.

To install the RichTextEditor component package, use the following command.

```
`bash
```

```
npm install @syncfusion/ej2-react-richtexteditor --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-react-richtexteditor
```

```
,
```

Import the Syncfusion CSS styles

After installing the Syncfusion component packages in the application, import the required themes based on the components used.

The Syncfusion React component comes with built-in [themes](#), which are available in installed packages. It is quite simple to adapt the Syncfusion React components based on the application style by referring to any of the built-in themes. Import the **Material** theme for the Rich Text Editor component.

Import the CSS styles for the Rich Text Editor component and its dependencies in the `src/App.css` file.

```
`css
```

```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-icons/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
```

```
,
```

Check out the [Themes topic](#) to know more about built-in themes and different ways to refer to themes in React applications.

Add Rich Text Editor component to the application

Initialize from React element

Start adding the required components to the application. Add the Rich Text Editor component in the `src/App.js` or `src/App.tsx` file using the following code.

- Before adding the Rich Text Editor component to the markup, import the Rich Text Editor component in the `src/App.js` or `src/App.tsx` file.

```
`bash
```

```
import { RichTextEditorComponent } from '@syncfusion/ej2-react-richtexteditor';
```

```
`
```

- Then, add the Rich Text Editor component in the application using the following code sample.

```
`ts
```

```
/
```

- Initilaize Rich Text Editor from React element

```
*/
```

```
import { HtmlEditor, Image, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
```

```
import * as React from 'react';
```

```
import './App.css';
```

```
function App() {
```

```
  return (
```

```
    <RichTextEditorComponent>
```

```
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides the best user experience to create and update the content. Users can format their content using standard toolbar commands.</p>
```

```
    <p><b>Key features:</b></p>
```

```
    <ul>
```

```
    <li>
```

```
    <p>Provides <IFRAME> and <DIV> modes</p>
```

```
    </li>
```

```
    <li>
```

```
    <p>Capable of handling markdown editing.</p>
```

```
</li>
<li>
<p>Contains a modular library to load the necessary functionality on demand.</p>
</li>
<li>
<p>Provides a fully customizable toolbar.</p>
</li>
<li>
<p>Provides HTML view to edit the source directly for developers.</p>
</li>
<li>
<p>Supports third-party library integration.</p>
</li>
<li>
<p>Allows preview of modified content before saving it.</p>
</li>
<li>
<p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
</li>
<li>
<p>Contains undo/redo manager.</p>
</li>
<li>
<p>Creates bulleted and numbered lists.</p>
</li>
</ul>
<Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
export default App;
`
```

Initialize from `<IFRAME>` element

The Rich Text Editor's content is placed in an iframe and isolated from the rest of the page.

Initialize the Rich Text Editor on div element and set the enable field of `iframeSettings` property to true.

Place the following Rich Text Editor code in the `app.tsx`.

```
`ts
/

    • Rich Text Editor Iframe Samples

*/

import { HtmlEditor, Image, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
const iframeSetting: object = { enable: true };
return (
<RichTextEditorComponent iframeSettings={iframeSetting}>
<p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides
the best user experience to create and update the content. Users can format their content using
standard toolbar commands.</p>
<p><b>Key features:</b></p>
<ul>
<li>
<p>Provides <IFRAME> and <DIV> modes</p>
</li>
<li>
<p>Capable of handling markdown editing.</p>
</li>
<li>
<p>Contains a modular library to load the necessary functionality on demand.</p>
</li>
<li>
<p>Provides a fully customizable toolbar.</p>
</li>
<li>
```

```

<p>Provides HTML view to edit the source directly for developers.</p>
</li>
<li>
<p>Supports third-party library integration.</p>
</li>
<li>
<p>Allows preview of modified content before saving it.</p>
</li>
<li>
<p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
</li>
<li>
<p>Contains undo/redo manager.</p>
</li>
<li>
<p>Creates bulleted and numbered lists.</p>
</li>
</ul>
<Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
export default App;
`

```

Module Injection

To create Rich Text Editor with additional features, inject the required modules. The following modules are used to extend Rich Text Editor's basic functionality.

- **Toolbar** - Inject this module to use Toolbar feature.
- **Link** - Inject this module to use link feature in Rich Text Editor.
- **Image** - Inject this module to use image feature in Rich Text Editor.
- **Table** - Inject this module to use table feature in Rich Text Editor.
- **Count** - Inject this module to use character count in Rich Text Editor.
- **HtmlEditor** - Inject this module to use Rich Text Editor as html editor.
- **MarkdownEditor** - Inject this module to use Rich Text Editor as markdown editor.
- **QuickToolbar** - Inject this module to use quick toolbar feature for the target element.

- **Resize** - Inject this module to use resize feature in Rich Text Editor.
- **FileManager** - Inject this module to use file browser feature in Rich Text Editor.
- **PasteCleanup** - Inject this module to use paste cleanup feature in Rich Text Editor.

These modules should be inject by using services.

Run the application

All are set. Now, run the application using the following command.

```
`bash
```

```
npm start
```

```
`
```

or

```
`bash
```

```
yarn start
```

```
`
```

The output will appear as follows.

APP.JSX

```
/**
 * Rich Text Editor - Getting Started Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  const toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  const quickToolbarSettings = {
    image: ['Replace', 'Align', 'Caption', 'Remove', 'InsertLink',
      'OpenImageLink', '-', 'EditImageLink', 'RemoveImageLink', 'Display',
      'AltText', 'Dimension'],
    link: ['Open', 'Edit', 'UnLink']
  };
  return (
    <RichTextEditorComponent height={450}
      toolbarSettings={toolbarSettings}
      quickToolbarSettings={quickToolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
```



```

        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
    </RichTextEditorComponent>;
  }
  export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Getting Started Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  const toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']

```

```

    }
    const quickToolbarSettings: object = {
      image: ['Replace', 'Align', 'Caption', 'Remove', 'InsertLink',
'OpenImageLink', '-', 'EditImageLink', 'RemoveImageLink', 'Display',
'AltText', 'Dimension'],
      link: ['Open', 'Edit', 'UnLink']
    }
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
quickToolbarSettings={quickToolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}
export default App;

```

Configure the Toolbar

Configure the toolbar with the tools using items field of the toolbarSettings property as your application requires.

APP.JSX

```
/**
 * Rich Text Editor - Toolbar Config Sample
 */
import { HtmlEditor, Inject, RichTextEditorComponent, Toolbar } from
'syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  const toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings}>
  <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
  Users can format their content using standard toolbar commands.</p>
  <p><b>Key features:</b></p>
  <ul>
    <li>
      <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
    </li>
    <li>
      <p>Capable of handling markdown editing.</p>
    </li>
    <li>
      <p>Contains a modular library to load the necessary functionality
on demand.</p>
    </li>
    <li>
      <p>Provides a fully customizable toolbar.</p>
    </li>
    <li>
      <p>Provides HTML view to edit the source directly for
developers.</p>
    </li>
    <li>
      <p>Supports third-party library integration.</p>
    </li>
    <li>
      <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
      <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
    </li>
  </ul>

```

```

        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, HtmlEditor]}/>
  </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Toolbar Config Sample
 */
import { HtmlEditor, Inject, RichTextEditorComponent, Toolbar } from
'@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  const toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>

```

```

        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, HtmlEditor]} />
    </RichTextEditorComponent>
  );
}
export default App;

```

The `|` and `-` can insert a vertical and horizontal separator lines in the toolbar.

Retrieve the formatted content

To retrieve the editor contents, use the `value` property of Rich Text Editor.

```

`ts
const rteValue: string = this.rteObj.value;
`

```

Or you can use the public method `getContent` to retrieve the RTE content.

```

`ts
const rteValue: string = this.rteObj.getContent();
`

```

To fetch the RichTextEditor's text content, use the `textContent` property of RTE `EditPanel`.

```

`ts
const rteValue: string = this.rteObj.contentModule.getEditPanel().textContent;
`

```

Insert images and links

The image module inserts an image into Rich Text Editor's content area, and the link module links external resources such as website URLs, to selected text in the Rich Text Editor's content, respectively.

The link inject module adds a link icon to the toolbar and the image inject module adds an image icon to the toolbar.

Specifies the items to be rendered in the quick toolbar based on the target element such image, link, and text element. The quick toolbar opens to customize the element by clicking the target element.

APP.JSX

```

/**
 * Rich Text Editor - Insert Image Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  const toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  const quickToolbarSettings = {
    image: ['Replace', 'Align', 'Caption', 'Remove', 'InsertLink',
'OpenImageLink', '-', 'EditImageLink', 'RemoveImageLink', 'Display',
'AltText', 'Dimension']
  };
  return (
    <RichTextEditorComponent height={450}
      toolbarSettings={toolbarSettings}
      quickToolbarSettings={quickToolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
          </li>
        </ul>
    </RichTextEditorComponent>
  );
}

```

```

        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
    </RichTextEditorComponent>);
  }
  export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Insert Image Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  const toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  }
  const quickToolbarSettings: object = {
    image: ['Replace', 'Align', 'Caption', 'Remove', 'InsertLink',
'OpenImageLink', '-', 'EditImageLink', 'RemoveImageLink', 'Display',
'AltText', 'Dimension']
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
quickToolbarSettings={quickToolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>

```

```

        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
  </RichTextEditorComponent>
);
}
export default App;

```

You can refer to our [React Rich Text Editor](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Rich Text Editor example](#) that shows how to render the rich text editor tools.

Editor modes in React Rich text editor component

The Rich Text Editor component used to create and edit the content and return valid HTML markup or markdown (MD) of the content. It supports the following two editing formation.

- HTML editor
- Markdown editor

HTML editor

Rich Text Editor is a WYSIWYG editing control for formatting the word content as HTML.

The HTML editing mode is the default mode in Rich Text Editor to format the content through the available toolbar items to return the valid HTML markup. Set the [editorMode](#) property as **HTML**.

To create Rich Text Editor with HTML editing feature, inject the **HtmlEditor** module to the RTE using the **RichTextEditor.Inject(HtmlEditor)** method.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - HTMLEditor Sample

```



```

*/
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  render() {
    return (<RichTextEditorComponent height={450}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>
    </RichTextEditorComponent>);
  }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - HTMLEditor Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}> {
  public render() {
    return (
      <RichTextEditorComponent height={450}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
          </li>
          <li>
            <p>Contains undo/redo manager.</p>
          </li>
          <li>
            <p>Creates bulleted and numbered lists.</p>
          </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
      </RichTextEditorComponent>
    );
  }
}

```

```
export default App;
```

[Functional-component]

APP.JSX

```
/**
 * Rich Text Editor - HTMLEditor Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  return (<RichTextEditorComponent height={450}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
  </RichTextEditorComponent>
);
```

```

    </RichTextEditorComponent>);
  }
  export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - HTMLEditor Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  return (
    <RichTextEditorComponent height={450}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
    </RichTextEditorComponent>
  );
}

```

```

        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
    />
    </RichTextEditorComponent>
  );
}
export default App;

```

Markdown editor

Set the [editorMode](#) property as **Markdown**, to create or edit the content and apply formatting to view markdown formatted content.

The third-party library such as **Marked** or any other library is used to convert markdown into HTML content.

- Supported tags are **h6, h5, h4, h3, h2, h1, blockquote, pre, p, OL, and UL.**
- Supported selection tags are **Bold, Italic, StrikeThrough, InlineCode, SubScript, SuperScript, UpperCase, and LowerCase.**

[Class-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - MarkdownEditor Sample
 */
import { createElement } from '@syncfusion/ej2-base';
import { Image, Inject, Link, MarkdownEditor, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as Marked from 'marked';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  // Rich Text Editor items list
  items = ['Bold', 'Italic', 'StrikeThrough', '|',
    'Formats', 'OrderedList', 'UnorderedList', '|',
    'CreateLink', 'Image', '|',
    {
      template: '<button id="preview-code" class="e-tbar-btn e-control e-btn e-icon-btn">' +
        '<span class="e-btn-icon e-md-preview e-icons"></span></button>',
      tooltipText: 'Preview',
    }, '|', 'Undo', 'Redo'];
  textArea;
  mdsSource;
  mdPreview;
  // Rich Text Editor ToolbarSettings
  toolbarSettings = {
    items: this.items
  };
  // set the value to Rich Text Editor
  template() {

```

```

    return (<div>
      The sample is added to showcase **markdown editing**.
      Type or edit the content and apply formatting to view markdown formatted
      content.
      We can add our own custom formation syntax for the Markdown formation,
      [sample link] (https://ej2.syncfusion.com/home/).
      The third-party library <b>Marked</b> is used in this sample to convert
      markdown into HTML content.
    </div>);
  }
;
  markDownConversion() {
    if (this.mdsource.classList.contains('e-active')) {
      const id = this.rteObj.getID() + 'html-view';
      const htmlPreview = this.rteObj.element.querySelector('#' + id);
      htmlPreview.innerHTML =
Marked.marked(this.rteObj.contentModule.getEditPanel().value);
    }
  }
  fullPreview() {
    const id = this.rteObj.getID() + 'html-preview';
    let htmlPreview = this.rteObj.element.querySelector('#' + id);
    if (this.mdsource.classList.contains('e-active')) {
      this.mdsource.classList.remove('e-active');
      this.mdsource.parentElement.title = 'Preview';
      this.textArea.style.display = 'block';
      htmlPreview.style.display = 'none';
    }
    else {
      this.mdsource.classList.add('e-active');
      if (!htmlPreview) {
        htmlPreview = createElement('div', { className: 'e-content e-
pre-source' });
        htmlPreview.id = id;
        this.textArea.parentNode.appendChild(htmlPreview);
      }
      this.textArea.style.display = 'none';
      htmlPreview.style.display = 'block';
      htmlPreview.innerHTML =
Marked.marked(this.rteObj.contentModule.getEditPanel().value);
      this.mdsource.parentElement.title = 'Code View';
    }
  }
  rendereComplete() {
    this.textArea = this.rteObj.contentModule.getEditPanel();
    this.textArea.addEventListener('keyup', (e) => {
      this.markDownConversion();
    });
    this.mdsource = document.getElementById('preview-code');
    this.mdsource.addEventListener('click', (e) => {
      this.fullPreview();
      if (e.currentTarget.classList.contains('e-active')) {
        this.rteObj.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', 'OrderedList',
'UnorderedList', 'CreateLink', 'Image', 'Formats',
'Undo', 'Redo']);
      }
    });
  }
}

```

```

        else {
            this.rteObj.enableToolbarItem(['Bold', 'Italic',
            'StrikeThrough', 'OrderedList',
            'UnorderedList', 'CreateLink', 'Image', 'Formats',
            'Undo', 'Redo']);
        }
    });
}
render() {
    return (<RichTextEditorComponent id="markdownRTE"
ref={(richtexteditor) => { this.rteObj = richtexteditor; }}
editorMode='Markdown' height='250px' valueTemplate={this.template}
toolbarSettings={this.toolbarSettings}
created={this.renderComplete.bind(this)}>
        <Inject services={ [MarkdownEditor, Toolbar, Image, Link,
QuickToolbar]} />
        </RichTextEditorComponent>);
}
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - MarkdownEditor Sample
 */
import { createElement, KeyboardEventArgs } from '@syncfusion/ej2-base';
import { Image, Inject, Link, MarkdownEditor, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    public rteObj: RichTextEditorComponent;
    // Rich Text Editor items list
    public items: object = ['Bold', 'Italic', 'StrikeThrough', '|',
'Formats', 'OrderedList', 'UnorderedList', '|',
'CreateLink', 'Image', '|',
{
        template: '<button id="preview-code" class="e-tbar-btn e-control e-
btn e-icon-btn">' +
        '<span class="e-btn-icon e-md-preview e-icons"></span></button>',
        tooltipText: 'Preview',
    }, '|', 'Undo', 'Redo'];
    public textArea: HTMLTextAreaElement;
    public mdsSource: any;
    public mdPreview: HTMLElement;
    // Rich Text Editor ToolbarSettings
    public toolbarSettings: object = {
        items: this.items
    };
    // set the value to Rich Text Editor
    public template(): JSX.Element {
        return (<div>
            The sample is added to showcase **markdown editing**.

```

Type or edit the content and apply formatting to view markdown formatted content.

We can **add** our own custom formation syntax **for** the Markdown formation, [sample link] (<https://ej2.syncfusion.com/home/>).

The third-party library **Marked** **is** used **in this** sample to convert markdown **into** HTML content.

```

    </div>);
};

    public markDownConversion(): void {
        if (this.mdsSource.classList.contains('e-active')) {
            const id: string = this.rteObj.getID() + 'html-view';
            const htmlPreview: HTMLElement =
this.rteObj.element.querySelector('#' + id) as any;
            htmlPreview.innerHTML = marked(((this.rteObj as
any).contentModule.getEditPanel() as HTMLTextAreaElement).value);
        }
    }

    public fullPreview(): void {
        const id: string = this.rteObj.getID() + 'html-preview';
        let htmlPreview: HTMLElement = (this.rteObj as
any).element.querySelector('#' + id);
        if (this.mdsSource.classList.contains('e-active')) {
            this.mdsSource.classList.remove('e-active');
            this.mdsSource.parentElement.title = 'Preview';
            this.textArea.style.display = 'block';
            htmlPreview.style.display = 'none';
        } else {
            this.mdsSource.classList.add('e-active');
            if (!htmlPreview) {
                htmlPreview = createElement('div', { className: 'e-content e-
pre-source' });
                htmlPreview.id = id;
                (this.textArea as any).parentNode.appendChild(htmlPreview);
            }
            this.textArea.style.display = 'none';
            htmlPreview.style.display = 'block';
            htmlPreview.innerHTML = marked(((this.rteObj as
any).contentModule.getEditPanel() as HTMLTextAreaElement).value);
            this.mdsSource.parentElement.title = 'Code View';
        }
    }

    public rendereComplete(): void {
        this.textArea = (this.rteObj as any).contentModule.getEditPanel() as
HTMLTextAreaElement;
        this.textArea.addEventListener('keyup', (e: KeyboardEventArgs) => {
            this.markDownConversion();
        });
        this.mdsSource = document.getElementById('preview-code');
        this.mdsSource.addEventListener('click', (e: MouseEvent) => {
            this.fullPreview();
            if ((e.currentTarget as HTMLElement).classList.contains('e-
active')) {
                this.rteObj.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', 'OrderedList',
'UnorderedList', 'CreateLink', 'Image', 'Formats', 'Undo',
'Redo']);
            } else {

```



```

        this.rteObj.enableToolbarItem(['Bold', 'Italic',
'StrikeThrough', 'OrderedList',
        'UnorderedList', 'CreateLink', 'Image', 'Formats', 'Undo',
'Redo']);
    }
    });
}
public render() {
    return (
        <RichTextEditorComponent id="markdownRTE" ref={(richtexteditor) =>
{ this.rteObj = richtexteditor! }} editorMode='Markdown'
        height='250px' valueTemplate={this.template}
toolbarSettings={this.toolbarSettings}
created={this.renderComplete.bind(this)}>
        <Inject services={[MarkdownEditor, Toolbar, Image, Link,
QuickToolbar]} />
        </RichTextEditorComponent>
    );
}
}
export default App;
{% endraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - MarkdownEditor Sample
 */
import { createElement } from '@syncfusion/ej2-base';
import { Image, Inject, Link, MarkdownEditor, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as Marked from 'marked';
import * as React from 'react';
function App() {
    let rteObj;
    // Rich Text Editor items list
    let items = ['Bold', 'Italic', 'StrikeThrough', '|',
        'Formats', 'OrderedList', 'UnorderedList', '|',
        'CreateLink', 'Image', '|',
        {
            template: '<button id="preview-code" class="e-tbar-btn e-control
e-btn e-icon-btn">' +
                '<span class="e-btn-icon e-md-preview e-
icons"></span></button>',
            tooltipText: 'Preview',
        }, '|', 'Undo', 'Redo'];
    let textArea;
    let mdsSource;
    let mdPreview;
    // Rich Text Editor ToolbarSettings
    let toolbarSettings = {
        items: items
    }
}

```

```

};
// set the value to Rich Text Editor
function template() {
    return (<div>
        The sample is added to showcase **markdown editing**. Type or edit
        the content and apply formatting to view markdown formatted content. We can
        add our own custom formation syntax for the Markdown formation, [sample
        link] (https://ej2.syncfusion.com/home/).
        The third-party library <b>Marked</b> is used in this sample to
        convert markdown into HTML content.
    </div>);
}
;
function markDownConversion() {
    if (mdsource.classList.contains('e-active')) {
        const id = rteObj.getID() + 'html-view';
        const htmlPreview = rteObj.element.querySelector('#' + id);
        htmlPreview.innerHTML =
Marked.marked(rteObj.contentModule.getEditPanel().value);
    }
}
function fullPreview() {
    const id = rteObj.getID() + 'html-preview';
    let htmlPreview = rteObj.element.querySelector('#' + id);
    if (mdsource.classList.contains('e-active')) {
        mdsource.classList.remove('e-active');
        mdsource.parentElement.title = 'Preview';
        textArea.style.display = 'block';
        htmlPreview.style.display = 'none';
    }
    else {
        mdsource.classList.add('e-active');
        if (!htmlPreview) {
            htmlPreview = createElement('div', { className: 'e-content e-
pre-source' });
            htmlPreview.id = id;
            textArea.parentNode.appendChild(htmlPreview);
        }
        textArea.style.display = 'none';
        htmlPreview.style.display = 'block';
        htmlPreview.innerHTML =
Marked.marked(rteObj.contentModule.getEditPanel().value);
        mdsource.parentElement.title = 'Code View';
    }
}
function rendereComplete() {
    textArea = rteObj.contentModule.getEditPanel();
    textArea.addEventListener('keyup', (e) => {
        markDownConversion();
    });
    mdsource = document.getElementById('preview-code');
    mdsource.addEventListener('click', (e) => {
        fullPreview();
        if (e.currentTarget.classList.contains('e-active')) {
            rteObj.disableToolbarItem(['Bold', 'Italic', 'StrikeThrough',
'OrderedList',

```

```

        'UnorderedList', 'CreateLink', 'Image', 'Formats',
        'Undo', 'Redo']]);
    }
    else {
        rteObj.enableToolbarItem(['Bold', 'Italic', 'StrikeThrough',
        'OrderedList',
        'UnorderedList', 'CreateLink', 'Image', 'Formats',
        'Undo', 'Redo']]);
    }
    });
}
return (<RichTextEditorComponent id="markdownRTE" ref={(richtexteditor)
=> { rteObj = richtexteditor; }} editorMode='Markdown' height='250px'
valueTemplate={template} created={rendereComplete}
toolbarSettings={toolbarSettings}>
    <Inject services={[MarkdownEditor, Toolbar, Image, Link,
QuickToolbar]}/>
    </RichTextEditorComponent>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - MarkdownEditor Sample
 */
import { createElement, KeyboardEventArgs } from '@syncfusion/ej2-base';
import { Image, Inject, Link, MarkdownEditor, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    let rteObj: RichTextEditorComponent;
    // Rich Text Editor items list
    let items: object = ['Bold', 'Italic', 'StrikeThrough', '|',
    'Formats', 'OrderedList', 'UnorderedList', '|',
    'CreateLink', 'Image', '|',
    {
        template: '<button id="preview-code" class="e-tbar-btn e-control e-btn
e-icon-btn">' +
        '<span class="e-btn-icon e-md-preview e-icons"></span></button>',
        tooltipText: 'Preview',
    }, '|', 'Undo', 'Redo'];
    let textArea: HTMLTextAreaElement;
    let mdsource: any;
    let mdPreview: HTMLElement;
    // Rich Text Editor ToolbarSettings
    let toolbarSettings: object = {
        items: items
    };
    // set the value to Rich Text Editor
    function template(): JSX.Element {
        return (
            <div>

```

The sample **is** added to showcase ****markdown editing****. Type or edit the content and apply formatting to view markdown formatted content. We can **add** our own custom formation syntax **for** the Markdown formation, [sample link] (<https://ej2.syncfusion.com/home/>).

The third-party library **Marked** **is** used **in this** sample to convert markdown **into** HTML content.

```

    </div>
  );
};
function markDownConversion(): void {
  if (mdsource.classList.contains('e-active')) {
    const id: string = rteObj.getID() + 'html-view';
    const htmlPreview: HTMLElement = rteObj.element.querySelector('#' +
id) as any;
    htmlPreview.innerHTML = marked(((rteObj as
any).contentModule.getEditPanel() as HTMLTextAreaElement).value);
  }
}
function fullPreview(): void {
  const id: string = rteObj.getID() + 'html-preview';
  let htmlPreview: HTMLElement = (rteObj as
any).element.querySelector('#' + id);
  if (mdsource.classList.contains('e-active')) {
    mdsource.classList.remove('e-active');
    mdsource.parentElement.title = 'Preview';
    textArea.style.display = 'block';
    htmlPreview.style.display = 'none';
  } else {
    mdsource.classList.add('e-active');
    if (!htmlPreview) {
      htmlPreview = createElement('div', { className: 'e-content e-
pre-source' });
      htmlPreview.id = id;
      (textArea as any).parentNode.appendChild(htmlPreview);
    }
    textArea.style.display = 'none';
    htmlPreview.style.display = 'block';
    htmlPreview.innerHTML = marked(((rteObj as
any).contentModule.getEditPanel() as HTMLTextAreaElement).value);
    mdsource.parentElement.title = 'Code View';
  }
}

function rendereComplete(): void {
  textArea = (rteObj as any).contentModule.getEditPanel() as
HTMLTextAreaElement;
  textArea.addEventListener('keyup', (e: KeyboardEventArgs) => {
    markDownConversion();
  });
  mdsource = document.getElementById('preview-code');
  mdsource.addEventListener('click', (e: MouseEvent) => {
    fullPreview();
    if ((e.currentTarget as HTMLElement).classList.contains('e-
active')) {
      rteObj.disableToolbarItem(['Bold', 'Italic', 'StrikeThrough',
'OrderedList',

```

```

        'UnorderedList', 'CreateLink', 'Image', 'Formats', 'Undo',
        'Redo']]);
    } else {
        rteObj.enableToolbarItem(['Bold', 'Italic', 'StrikeThrough',
        'OrderedList',
        'UnorderedList', 'CreateLink', 'Image', 'Formats', 'Undo',
        'Redo']]);
    }
    });
}
return (
    <RichTextEditorComponent id="markdownRTE" ref={(richtexteditor) =>
    {rteObj = richtexteditor! }} editorMode='Markdown'
    height='250px' valueTemplate={template} created={rendereComplete}
    toolbarSettings={toolbarSettings} >
        <Inject services={[MarkdownEditor, Toolbar, Image, Link,
        QuickToolbar]} />
        </RichTextEditorComponent>
    );
}
export default App;
{% enddraw %}

```

To create Rich Text Editor with Markdown editing feature, inject the `MarkdownEditor` module to the RTE using the `RichTextEditor.Inject(MarkdownEditor)` method.

For further details on Markdown editing, refer to the [Markdown](#) section.

Toolbar in React Rich text editor component

The Rich Text Editor toolbar contains a collection of tools such as bold, italic, and text alignment buttons that are used to format the content. However, in most integrations, you can customize the toolbar configurations easily to suit your needs.

To create Rich Text Editor with Markdown editing feature, inject the toolbar module to the RTE using the `RichTextEditor.Inject(Toolbar)` method.

The Rich Text Editor allows you to configure the different types of toolbar using the `toolbarSettings.type` property. The types of toolbar are:

- Expand
- MultiRow

Expand Toolbar

The default mode of `toolbarSettings.type` is Expand to hide the overflowing items in the next row. By clicking the expand arrow, view the overflowing toolbar items.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - Expand Toolbar Sample
 */

```

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo'
    ],
    type: 'Expand'
  };
  render() {
    return (<RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>

```

```

        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
    </RichTextEditorComponent>);
  }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Expand Toolbar Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  public toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo'
    ],
    type: 'Expand'
  }
  public render() {
    return (
      <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>

```

```

        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Expand Toolbar Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo'
],
    type: 'Expand'
  };
  return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>

```



```

    <li>
      <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
    </li>
    <li>
      <p>Capable of handling markdown editing.</p>
    </li>
    <li>
      <p>Contains a modular library to load the necessary functionality
on demand.</p>
    </li>
    <li>
      <p>Provides a fully customizable toolbar.</p>
    </li>
    <li>
      <p>Provides HTML view to edit the source directly for
developers.</p>
    </li>
    <li>
      <p>Supports third-party library integration.</p>
    </li>
    <li>
      <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
      <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
    </li>
    <li>
      <p>Contains undo/redo manager.</p>
    </li>
    <li>
      <p>Creates bulleted and numbered lists.</p>
    </li>
  </ul>
  <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
</RichTextEditorComponent>;
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Expand Toolbar Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };

```

```

    ],
    type: 'Expand'
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides the best user experience to create and update the content.
        Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}
export default App;

```

MultiRow toolbar

Set the [toolbarSettings.type](#) as MultiRow to hide the overflowing items in the next row. All toolbar items are visible.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - MultiRow Toolbar Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo'
],
    type: 'MultiRow'
  };
  render() {
    return (
      <RichTextEditorComponent height={450}
        toolbarSettings={this.toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
          </li>
        </ul>
      </RichTextEditorComponent>
    );
  }
}

```

```

        <li>
            <p>Contains undo/redox manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
</RichTextEditorComponent>);
    }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - MultiRow Toolbar Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    private toolbarSettings: object = {
        items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
            'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
            'LowerCase', 'UpperCase', '|',
            'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
            'Outdent', 'Indent', '|',
            'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
            'SourceCode', 'FullScreen', '|', 'Undo', 'Redo'
        ],
        type: 'MultiRow'
    }
    public render() {
        return (
            <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}>
                <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
                Users can format their content using standard toolbar commands.</p>
                <p><b>Key features:</b></p>
                <ul>
                    <li>
                        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
                    </li>
                    <li>
                        <p>Capable of handling markdown editing.</p>
                    </li>
                    <li>
                        <p>Contains a modular library to load the necessary functionality
on demand.</p>
                    </li>
                    <li>
                        <p>Provides a fully customizable toolbar.</p>
                    </li>
                </ul>
            </RichTextEditorComponent>
        )
    }
}

```

```

        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - MultiRow Toolbar Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo'
],
    type: 'MultiRow'
  };
  return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings}>

```

```

    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
    get") editor that provides the best user experience to create and update the
    content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
  </RichTextEditorComponent>;
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - MultiRow Toolbar Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',

```

```

        'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
        'LowerCase', 'UpperCase', '|',
        'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
        'Outdent', 'Indent', '|',
        'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
        'SourceCode', 'FullScreen', '|', 'Undo', 'Redo'
    ],
    type: 'MultiRow'
}
return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
                <p>Capable of handling markdown editing.</p>
            </li>
            <li>
                <p>Contains a modular library to load the necessary functionality
on demand.</p>
            </li>
            <li>
                <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
                <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
                <p>Supports third-party library integration.</p>
            </li>
            <li>
                <p>Allows preview of modified content before saving it.</p>
            </li>
            <li>
                <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
            </li>
            <li>
                <p>Contains undo/redo manager.</p>
            </li>
            <li>
                <p>Creates bulleted and numbered lists.</p>
            </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
);
}
export default App;

```

Floating toolbar

By default, the toolbar is floating at the top of the Rich Text Editor on scrolling. It can be customized by specifying the offset of the floating toolbar from documents top position using [floatingToolbarOffset](#).

Enable or disable the floating toolbar using [enableFloating](#) of the toolbarSetting property.

[Class-component]

APP.JSX

```
{% raw %}
import { CheckBoxComponent } from '@syncfusion/ej2-react-buttons';
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  checkboxObj;
  rteObj;
  toolbarSettings = {
    enableFloating: false
  };
  onChange(args) {
    this.rteObj.toolbarSettings.enableFloating = args.checked;
    this.rteObj.dataBind();
  }
  render() {
    return (<div>
      <CheckBoxComponent checked={false} label='Enable Floating'
ref={(scope) => { this.checkboxObj = scope; }} change={this.onChange =
this.onChange.bind(this)} />
      <br />
      <br />
      <br />
      <RichTextEditorComponent ref={(scope) => { this.rteObj = scope; }}
height={450} toolbarSettings={this.toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary
functionality on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>

```



```

        <p>Provides HTML view to edit the source directly for
developers.</p>
    </li>
    <li>
        <p>Supports third-party library integration.</p>
    </li>
    <li>
        <p>Allows preview of modified content before saving
it.</p>
    </li>
    <li>
        <p>Handles images, hyperlinks, video, hyperlinks,
uploads, etc.</p>
    </li>
    <li>
        <p>Contains undo/redo manager.</p>
    </li>
    <li>
        <p>Creates bulleted and numbered lists.</p>
    </li>
</ul>
<Inject services={[Toolbar, Image, Link, HtmlEditor,
QuickToolbar]}/>
</RichTextEditorComponent>
</div>);
    }
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Floating Toolbar Sample
 */
import { ChangeEventArgs } from '@syncfusion/ej2-buttons';
import { CheckBoxComponent } from '@syncfusion/ej2-react-buttons';
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    public checkboxObj: CheckBoxComponent;
    public rteObj: RichTextEditorComponent;
    public toolbarSettings: object = {
        enableFloating: false
    }
    public onChange(args: ChangeEventArgs): void {
        this.rteObj.toolbarSettings.enableFloating = args.checked;
        this.rteObj.dataBind();
    }
    public render() {
        return (
            <div>

```

```

        <CheckBoxComponent checked={false} label='Enable Floating'
ref={ (scope) => { this.checkboxObj = scope! }} change={
this.onChange=this.onChange.bind(this) } />
        <br />
        <br />
        <br />
        <RichTextEditorComponent ref={ (scope) => { this.rteObj = scope! }}
height={450} toolbarSettings={this.toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
                <p>Capable of handling markdown editing.</p>
            </li>
            <li>
                <p>Contains a modular library to load the necessary
functionality on demand.</p>
            </li>
            <li>
                <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
                <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
                <p>Supports third-party library integration.</p>
            </li>
            <li>
                <p>Allows preview of modified content before saving
it.</p>
            </li>
            <li>
                <p>Handles images, hyperlinks, video, hyperlinks,
uploads, etc.</p>
            </li>
            <li>
                <p>Contains undo/redo manager.</p>
            </li>
            <li>
                <p>Creates bulleted and numbered lists.</p>
            </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor,
QuickToolbar]} />
    </RichTextEditorComponent>
</div>
    );
}
}
export default App;

```

```
{% endraw %}
```

[functional-component]

APP.JSX

```
{% raw %}
import { CheckBoxComponent } from '@syncfusion/ej2-react-buttons';
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let checkboxObj;
  let rteObj;
  let toolbarSettings = {
    enableFloating: false
  };
  function onChange(args) {
    rteObj.toolbarSettings.enableFloating = args.checked;
    rteObj.dataBind();
  }
  return (<div>
    <CheckBoxComponent checked={false} label='Enable Floating'
ref={ (scope) => { checkboxObj = scope; }} change={onChange.bind(this)} />
    <br />
    <br />
    <br />
    <RichTextEditorComponent ref={ (scope) => { rteObj = scope; }}
height={450} toolbarSettings={toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is
what you get") editor that provides the best user experience to create and
update the content. Users can format their content using standard toolbar
commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary
functionality on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
      </ul>
    </RichTextEditorComponent>
  </div>
  </App>
  </pre>

```

```

        <li>
            <p>Allows preview of modified content before saving
it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks,
uploads, etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor,
QuickToolbar]}/>
</RichTextEditorComponent>
</div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Floating Toolbar Sample
 */
import { ChangeEventArgs } from '@syncfusion/ej2-buttons';
import { CheckBoxComponent } from '@syncfusion/ej2-react-buttons';
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    let checkboxObj: CheckBoxComponent;
    let rteObj: RichTextEditorComponent;
    let toolbarSettings: object = {
        enableFloating: false
    }
    function onChange(args: ChangeEventArgs): void {
        rteObj.toolbarSettings.enableFloating = args.checked;
        rteObj.dataBind();
    }
    return (
        <div>
            <CheckBoxComponent checked={false} label='Enable Floating'
ref={(scope) => {checkboxObj = scope! }} change={onChange.bind(this)} />
            <br />
            <br />
            <br />
            <RichTextEditorComponent ref={(scope) => {rteObj = scope! }}
height={450} toolbarSettings={toolbarSettings}>
                <p>The Rich Text Editor component is WYSIWYG ("what you see is
what you get") editor that provides the best user experience to create and

```

```















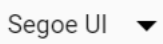

update the content. Users can format their content using standard toolbar
commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary
functionality on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving
it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks,
uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor,
QuickToolbar]} />
  </RichTextEditorComponent>
</div>
);
}
export default App;
{% endraw %}


```


Toolbar items

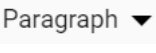
The following table lists the tools available in the toolbar.


Name	Icons	Summary	Initialization
---	---	---	---


- | Undo |  | Allows to undo the actions. | toolbarSettings: {
 items: ['Undo']
 } |
- | Redo |  | Allows to redo the actions. | toolbarSettings: {
 items: ['Redo']
 } |
- | Alignment |  | Align the content with left, center, and right margin. | toolbarSettings: {
 items: ['Alignments']
 } |
- | OrderedList |  | Create a new list item (numbered). | toolbarSettings: {
 items: ['OrderedList']
 } |
- | UnorderedList |  | Create a new list item (bulleted). | toolbarSettings: {
 items: ['UnorderedList']
 } |
- | Indent |  | Allows to increase the indent level of the content. | toolbarSettings: {
 items: ['Indent']
 } |
- | Outdent |  | Allows to decrease the indent level of the content. | toolbarSettings: {
 items: ['Outdent']
 } |
- | Hyperlink |  | Creates a hyperlink to a text or image to a specific location in the content. | toolbarSettings: {
 items: ['CreateLink']
 } |
- | Images |  | Inserts an image from an online source or local computer. | toolbarSettings: {
 items: ['Image']
 } |
- | LowerCase |  | Change the case of selected text to lower in the content. | toolbarSettings: {
 items: ['LowerCase']
 } |
- | UpperCase |  | Change the case of selected text to upper in the content. | toolbarSettings: {
 items: ['UpperCase']
 } |
- | SubScript |  | Makes the selected text as subscript (lower). | toolbarSettings: {
 items: ['SubScript']
 } |
- | SuperScript |  | Makes the selected text as superscript (higher). | toolbarSettings: {
 items: ['SuperScript']
 } |
- | Print |  | Allows to print the editor content. | toolbarSettings: {
 items: ['Print']
 } |
- | FontName |  | Defines the fonts that appear under the Font Family DropDownList from the Rich Text Editor's toolbar. | toolbarSettings: {
 items: ['FontName']
 } |
- | FontSize |  | Defines the font sizes that appear under the Font Size DropDownList from the Rich Text Editor's toolbar. | toolbarSettings: {
 items: ['FontSize']
 } |


| FontColor |  | Specifies an array of colors can be used in the colors pop-up for font color. |
 toolbarSettings: {
 items: ['FontColor']
 } |


| BackgroundColor |  | Specifies an array of colors can be used in the colors pop-up for background color. | toolbarSettings: {
 items: ['BackgroundColor']
 } |


| Format |  | An object with the options that will appear in the paragraph format drop-down from the toolbar. | toolbarSettings: {
 items: ['Formats']
 } |


| StrikeThrough |  | Apply double line strike through formatting for the selected text. |
 toolbarSettings: {
 items: ['StrikeThrough']
 } |


| ClearFormat |  | The clear format tool is used to remove all formatting styles (such as bold, italic, underline, color, superscript, subscript, and more) from currently selected text. As a result, all the formatting text will be cleared and return to its default formatting styles. | toolbarSettings: {
 items: ['ClearFormat']
 } |


| FullScreen |  | Stretches the editor to the maximum width and height of the browser window. |
 toolbarSettings: {
 items: ['FullScreen']
 } |


| SourceCode |  | The RichTextBox includes the ability for users to directly edit the HTML code via Source View. If you made any modification in source view directly, synchronize with design view. |
 toolbarSettings: {
 items: ['SourceCode']
 } |


| NumberFormatList |  | Allows to create list items with various list style types(numbered).| toolbarSettings: {
 items: ['NumberFormatList']
 } |


| BulletFormatList |  | Allows to create list items with various list style types(bulleted).| toolbarSettings: {
 items: ['BulletFormatList']
 } |


| JustifyLeft |  | Allows each line to begin at the same distance from the editor's left-hand side. |
 toolbarSettings: {
 items: ['JustifyLeft']
 } |


| JustifyCenter |  | There is an even space on each side of each line since the text is not aligned to the left or right margins. | toolbarSettings: {
 items: ['JustifyCenter']
 } |


| JustifyRight |  | Allows each line to end at the same distance from the editor's right-hand side. |
 toolbarSettings: {
 items: ['JustifyRight']
 } |


| JustifyFull |  | The text is aligned with both right and left margins. | toolbarSettings: {
 items: ['JustifyFull']
 } |


| Bold |  | Text that is thicker and darker than usual. | toolbarSettings: {
 items: ['Bold']
 } |


| Italic |  | Shows a text that is leaned to the right. | toolbarSettings: {
 items: ['Italic']
 } |


| Underline |  | The underline is added to the selected text. | toolbarSettings: {
 items: ['Underline']
 } |


| ClearAll |  | Removes all styles that have been applied to the selected text. | toolbarSettings: {
 items: ['ClearAll']
 } |


| Cut |  | Removes the text from its current location and places it into the clipboard. | toolbarSettings: {
 items: ['Cut']
 } |


| Copy |  | The selected item is copied and pasted into the clipboard. | toolbarSettings: {
 items: ['Copy']
 } |


| Paste |  | Allows you to insert a clipboard item into a specific location. | toolbarSettings: {
 items: ['Paste']
 } |


| OpenLink |  | To open the URL link that is attached to the selected text. | toolbarSettings: {
 items: ['OpenLink']
 } |


| EditLink |  | Allows you to change the URL that has been attached to a specific item. | toolbarSettings: {
 items: ['EditLink']
 } |


| CreateTable |  | Create a table with defined columns and rows. | toolbarSettings: {
 items: ['CreateTable']
 } |


| RemoveTable |  | Removes the selected table and its contents. | toolbarSettings: {
 items: ['TableRemove']
 } |


| Replace |  | Replace the selected image with another image. | toolbarSettings: {
 items: ['Replace']
 } |


| Align |  | The image can be aligned to the right, left, or center. | toolbarSettings: {
 items: ['Align']
 } |


| Remove |  | Allows to remove the selected image from the editor. | toolbarSettings: {
 items: ['Remove']
 } |


| OpenImageLink |  | Opens the link that is attached to the selected image. | toolbarSettings: {
 items: ['OpenImageLink']
 } |


 | EditImageLink | Allows to edit the link that is attached to the selected image. | toolbarSettings: {
 items: ['EditImageLink']
 } |


 | RemoveImageLink | Removes the link that is attached to the selected image. | toolbarSettings: {
 items: ['RemoveImageLink']
 } |


 | InsertLink | Allows users to add a link to a particular item. | toolbarSettings: {
 items: ['InsertLink']
 } |


 | Display | Allows you to choose whether an image should be shown inline or as a block. | toolbarSettings: {
 items: ['Display']
 } |


 | AltText | To display image description when an image on a Web page cannot be displayed. | toolbarSettings: {
 items: ['AltText']
 } |


 | Dimension | Allows you to customize the image's height and width. | toolbarSettings: {
 items: ['Dimension']
 } |


 | Maximize | Stretches the editor to the maximum width and height of the browser window. | toolbarSettings: {
 items: ['Maximize']
 } |


 | Minimize | Shrinks the editor to the default width and height. | toolbarSettings: {
 items: ['Minimize']
 } |


 | Preview | Allows to see how the editor's content looks in a browser. | toolbarSettings: {
 items: ['Preview']
 } |


 | InsertCode | Represents preformatted text which is to be presented exactly as written in the HTML file. | toolbarSettings: {
 items: ['InsertCode']
 } |


 | RemoveLink | Allows you to remove the applied link from the selected item. | toolbarSettings: {
 items: ['RemoveLink']
 } |


 | TableHeader | Allows you to add a table header. | toolbarSettings: {
 items: ['TableHeader']
 } |

 | TableColumns | Shows the dropdown to insert a column or delete the selected column. | toolbarSettings: {
 items: ['TableColumns']
 } |

 | TableRows | Shows the dropdown to insert a row or delete the selected row. | toolbarSettings: {
 items: ['TableRows']
 } |

| TableCellHorizontalAlign |  | Allows the table cell content to be aligned horizontally. |
 toolbarSettings: {
 items: ['TableCellHorizontalAlign']
 } |

| TableCellVerticalAlign |  | Allows the table cell content to be aligned vertically. |
 toolbarSettings: {
 items: ['TableCellVerticalAlign']
 } |

| TableEditProperties |  | Allows you to change the table width, padding, and cell spacing styles. |
 toolbarSettings: {
 items: ['TableEditProperties']
 } |

By default, tools will be arranged in the following order.

```
` javascript
items: ['Bold', 'Italic', 'Underline', '|', 'Formats', 'Alignments', 'OrderedList', 'UnorderedList', '|',
'CreateLink', 'Image', '|', 'SourceCode', 'Undo', 'Redo']
`
```

The tools order can be customized as our application requirement. If you are not specifying any tools order, the editor will create the toolbar with default items.

Custom tool

The Rich Text Editor allows you to configure your own commands to its toolbar using the [toolbarSettings](#) property. The command can be plain text, icon, or HTML template. The order and the group can also be defined where the command should be included. Bind the action to the command by getting its instance.

This sample shows how to add your own commands to the toolbar of the Rich Text Editor. The **Ω** command is added to insert special characters in the editor. By clicking the **Ω** command, it will show the special characters list, and then choose the character to be inserted in the editor.

The following code snippet illustrates custom tool with tooltip text which will be included in [items](#) field of the toolbarSettings property.

In the following sample, once Rich Text Editor control is [created](#), the concern event will be created; the Dialog component can be rendered and target as RTE content.

```
` javascript
{
template: '<button class="e-tbar-btn e-btn" tabindex="-1" id="custom_tbar" style="width:100%"><div
class="e-tbar-btn-text" style="font-weight: 500;"> Ω</div></button>',
undo: true,
click: this.onClick.bind(this),
tooltipText: 'Insert Symbol'
}
`
```

[Class-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - Custom Tool Sample
 */
import { DialogComponent } from '@syncfusion/ej2-react-popups';
import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  // set the value to Rich Text Editor
  template = `<div style="display:block;"><p style="margin-right:10px">The
custom command "insert special character" is configured as the last item of
the toolbar. Click on the command and choose the special character you want
to include from the popup.</p></div>`;
  selection = new NodeSelection();
  ranges;
  dialogObj;
  // Rich Text Editor items list
  items = ['Bold', 'Italic', 'Underline', '|', 'Formats', 'Alignments',
'OrderedList',
  'UnorderedList', '|', 'CreateLink', 'Image', '|', 'SourceCode',
  {
    template: '<button class="e-tbar-btn e-btn" tabindex="-1"
id="custom_tbar" style="width:100%"><div class="e-tbar-btn-text"
style="font-weight: 500;"> &#937;</div></button>',
    undo: true,
    click: this.onClick.bind(this),
    tooltipText: 'Insert Symbol'
  }, '|', 'Undo', 'Redo'
];
  // Rich Text Editor ToolbarSettings
  toolbarSettings = {
    items: this.items
  };
  dlgButtons = [
    { buttonModel: { content: "Insert", isPrimary: true }, click:
this.onInsert.bind(this) },
    { buttonModel: { content: 'Cancel' }, click: this.onCancel }
  ];
  header = 'Special Characters';
  target = document.getElementById('rteSection');
  height = 'auto';
  dialogCreate() {
    const dialogCtn = document.getElementById('rteSpecial_char');
    dialogCtn.onclick = (e) => {
      const target = e.target;
      const activeEle =
this.dialogObj.element.querySelector('.char_block.e-active');
      if (target.classList.contains('char_block')) {
        target.classList.add('e-active');
        if (activeEle) {
          activeEle.classList.remove('e-active');
        }
      }
    }
  };
}

```

```

    }
    onInsert() {
        const activeEle =
this.dialogObj.element.querySelector('.char_block.e-active');
        if (activeEle) {

this.ranges.insertNode(document.createTextNode(activeEle.textContent));
        }
        this.dialogOverlay();
    }
    onClick() {
        this.ranges = this.selection.getRange(document);
        this.dialogObj.width = this.rteObj.element.offsetWidth * 0.5;
        this.dialogObj.content = document.getElementById('rteSpecial_char');
        this.dialogObj.dataBind();
        this.dialogObj.show();
    }
    dialogOverlay() {
        const activeEle =
this.dialogObj.element.querySelector('.char_block.e-active');
        if (activeEle) {
            activeEle.classList.remove('e-active');
        }
        this.dialogObj.hide();
    }
    onCancel(e) {
        const activeEle = this.element.querySelector('.char_block.e-active');
        if (activeEle) {
            activeEle.classList.remove('e-active');
        }
        this.hide();
    }
    render() {
        const hideDiv = { display: "none" };
        return (<div className='control-pane'>
            <div className='control-section e-rte-custom-tbar-section'
id="rteCustomTool">
                <div className='rte-control-section' id='rteSection'>
                    <RichTextEditorComponent id="defaultRTE" ref={ (scope) => {
this.rteObj = scope; }} valueTemplate={this.template}
toolbarSettings={this.toolbarSettings}>
                        <Inject services={[HtmlEditor, Toolbar, Link, Image,
QuickToolbar]}/>
                    </RichTextEditorComponent>
                    <DialogComponent id='customTbarDlg' ref={ (scope) => {
this.dialogObj = scope; }} buttons={this.dlgButtons}
overlayClick={this.dialogOverlay} = this.dialogOverlay.bind(this)}
header={this.header} visible={false} showCloseIcon={false}
target={'#rteSection'} height={this.height} created={this.dialogCreate =
this.dialogCreate.bind(this)} isModal={true} cssClass={'e-rte-elements'}/>
                        <div id="customTbarDialog" style={hideDiv}>
                            <div id="rteSpecial_char">
                                <div className="char_block" title="&#94;">&#94;</div>
                                <div className="char_block" title="&#95;">&#95;</div>
                                <div className="char_block" title="&#96;">&#96;</div>
                                <div className="char_block" title="&#123;">&#123;</div>
                                <div className="char_block" title="&#124;">&#124;</div>

```

```

        <div className="char_block" title="#125;">#125;</div>
        <div className="char_block" title="#126;">#126;</div>
        <div className="char_block" title="#160;">#160;</div>
        <div className="char_block" title="#161;">#161;</div>
        <div className="char_block" title="#162;">#162;</div>
        <div className="char_block" title="#163;">#163;</div>
        <div className="char_block" title="#164;">#164;</div>
        <div className="char_block" title="#165;">#165;</div>
        <div className="char_block" title="#x20B9;">#x20B9;</div>
        <div className="char_block" title="#166;">#166;</div>
        <div className="char_block" title="#167;">#167;</div>
        <div className="char_block" title="#168;">#168;</div>
        <div className="char_block" title="#169;">#169;</div>
        <div className="char_block" title="#170;">#170;</div>
        <div className="char_block" title="#171;">#171;</div>
        <div className="char_block" title="#172;">#172;</div>
        <div className="char_block" title="#173;">#173;</div>
        <div className="char_block" title="#174;">#174;</div>
        <div className="char_block" title="#175;">#175;</div>
        <div className="char_block" title="#176;">#176;</div>
        <div className="char_block" title="#177;">#177;</div>
        <div className="char_block" title="#178;">#178;</div>
        <div className="char_block" title="#179;">#179;</div>
        <div className="char_block" title="#180;">#180;</div>
        <div className="char_block" title="#181;">#181;</div>
        <div className="char_block" title="#182;">#182;</div>
        <div className="char_block" title="#183;">#183;</div>
        <div className="char_block" title="#184;">#184;</div>
        <div className="char_block" title="#185;">#185;</div>
        <div className="char_block" title="#186;">#186;</div>
        <div className="char_block" title="#187;">#187;</div>
        <div className="char_block" title="#188;">#188;</div>
        <div className="char_block" title="#189;">#189;</div>
        <div className="char_block" title="#190;">#190;</div>
        <div className="char_block" title="#191;">#191;</div>
        <div className="char_block" title="#192;">#192;</div>
        <div className="char_block" title="#193;">#193;</div>
        <div className="char_block" title="#194;">#194;</div>
        <div className="char_block" title="#195;">#195;</div>
    </div>
</div>
</div>
</div>
</div>);
    }
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Custom Tool Sample
 */
import { DialogComponent } from '@syncfusion/ej2-react-popups';

```

```

import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{},{}> {
    public rteObj: RichTextEditorComponent;
    // set the value to Rich Text Editor
    public template: string = `<div style="display:block;"><p style="margin-
right:10px">The custom command "insert special character" is configured as
the last item of the toolbar. Click on the command and choose the special
character you want to include from the popup.</p></div>`;
    public selection: NodeSelection = new NodeSelection();
    public ranges: Range;
    public dialogObj: DialogComponent;
    // Rich Text Editor items list
    public items: object = ['Bold', 'Italic', 'Underline', '|', 'Formats',
'Alignments', 'OrderedList',
    'UnorderedList', '|', 'CreateLink', 'Image', '|', 'SourceCode',
    {
        template: '<button class="e-tbar-btn e-btn" tabindex="-1"
id="custom_tbar" style="width:100%"><div class="e-tbar-btn-text"
style="font-weight: 500;"> &#937;</div></button>',
        undo: true,
        click: this.onClick.bind(this),
        tooltipText: 'Insert Symbol'
    }, '|', 'Undo', 'Redo'
    ];
    // Rich Text Editor ToolbarSettings
    public toolbarSettings: object = {
        items: this.items
    };
    public dlgButtons: any = [
        { buttonModel: { content: "Insert", isPrimary: true }, click:
this.onInsert.bind(this) },
        { buttonModel: { content: 'Cancel' }, click: this.onCancel }
    ];
    public header: string = 'Special Characters';
    public target: Element = document.getElementById('rteSection') as any;
    public height: any = 'auto';
    public dialogCreate(): void {
        const dialogCtn: HTMLElement = document.getElementById('rteSpecial_char')
as HTMLElement;
        dialogCtn.onclick = (e: Event) => {
            const target: HTMLElement = e.target as HTMLElement;
            const activeEle: Element =
this.dialogObj.element.querySelector('.char_block.e-active') as HTMLElement;
            if (target.classList.contains('char_block')) {
                target.classList.add('e-active');
                if (activeEle) {
                    activeEle.classList.remove('e-active');
                }
            }
        };
    }
    public onInsert(): void {
        const activeEle: Element =
this.dialogObj.element.querySelector('.char_block.e-active') as any;

```

```

        if (activeEle) {
            this.ranges.insertNode(document.createTextNode(activeEle.textContent as
string));
        }
        this.dialogOverlay();
    }
    public onClick(): void {
        this.ranges = this.selection.getRange(document);
        this.dialogObj.width = (this.rteObj as any).element.offsetWidth * 0.5;
        this.dialogObj.content = document.getElementById('rteSpecial_char') as
HTMLInputElement;
        this.dialogObj.dataBind();
        this.dialogObj.show();
    }
    public dialogOverlay(): void {
        const activeEle: Element =
this.dialogObj.element.querySelector('.char_block.e-active') as HTMLInputElement;
        if (activeEle) {
            activeEle.classList.remove('e-active');
        }
        this.dialogObj.hide();
    }
    public onCancel(e: any): void {
        const activeEle: Element = (this as
any).element.querySelector('.char_block.e-active');
        if (activeEle) {
            activeEle.classList.remove('e-active');
        }
        (this as any).hide();
    }
    public render() {
        const hideDiv = { display: "none" };
        return (
            <div className='control-pane'>
                <div className='control-section e-rte-custom-tbar-section'
id="rteCustomTool">
                    <div className='rte-control-section' id='rteSection'>
                        <RichTextEditorComponent id="defaultRTE" ref={ (scope) => {
this.rteObj = scope! } }
                            valueTemplate={this.template}
toolbarSettings={this.toolbarSettings}>
                            <Inject services={[HtmlEditor, Toolbar, Link, Image,
QuickToolbar]} />
                        </RichTextEditorComponent>
                        <DialogComponent id='customTbarDlg' ref={ (scope) => {
this.dialogObj = scope! } }
                            buttons={this.dlgButtons} overlayClick={this.dialogOverlay =
this.dialogOverlay.bind(this)} header={this.header} visible={false}
                            showCloseIcon={false} target={'#rteSection'}
height={this.height} created={this.dialogCreate =
this.dialogCreate.bind(this)} isModal={true} cssClass={'e-rte-elements'} />
                        <div id="customTbarDialog" style={hideDiv}>
                            <div id="rteSpecial_char">
                                <div className="char_block" title="#94;">#94;</div>
                                <div className="char_block" title="#95;">#95;</div>
                                <div className="char_block" title="#96;">#96;</div>
                                <div className="char_block" title="#123;">#123;</div>

```

```

        <div className="char_block" title="#124;">#124;</div>
        <div className="char_block" title="#125;">#125;</div>
        <div className="char_block" title="#126;">#126;</div>
        <div className="char_block" title="#160;">#160;</div>
        <div className="char_block" title="#161;">#161;</div>
        <div className="char_block" title="#162;">#162;</div>
        <div className="char_block" title="#163;">#163;</div>
        <div className="char_block" title="#164;">#164;</div>
        <div className="char_block" title="#165;">#165;</div>
        <div className="char_block" title="#x20B9;">#x20B9;</div>
        <div className="char_block" title="#166;">#166;</div>
        <div className="char_block" title="#167;">#167;</div>
        <div className="char_block" title="#168;">#168;</div>
        <div className="char_block" title="#169;">#169;</div>
        <div className="char_block" title="#170;">#170;</div>
        <div className="char_block" title="#171;">#171;</div>
        <div className="char_block" title="#172;">#172;</div>
        <div className="char_block" title="#173;">#173;</div>
        <div className="char_block" title="#174;">#174;</div>
        <div className="char_block" title="#175;">#175;</div>
        <div className="char_block" title="#176;">#176;</div>
        <div className="char_block" title="#177;">#177;</div>
        <div className="char_block" title="#178;">#178;</div>
        <div className="char_block" title="#179;">#179;</div>
        <div className="char_block" title="#180;">#180;</div>
        <div className="char_block" title="#181;">#181;</div>
        <div className="char_block" title="#182;">#182;</div>
        <div className="char_block" title="#183;">#183;</div>
        <div className="char_block" title="#184;">#184;</div>
        <div className="char_block" title="#185;">#185;</div>
        <div className="char_block" title="#186;">#186;</div>
        <div className="char_block" title="#187;">#187;</div>
        <div className="char_block" title="#188;">#188;</div>
        <div className="char_block" title="#189;">#189;</div>
        <div className="char_block" title="#190;">#190;</div>
        <div className="char_block" title="#191;">#191;</div>
        <div className="char_block" title="#192;">#192;</div>
        <div className="char_block" title="#193;">#193;</div>
        <div className="char_block" title="#194;">#194;</div>
        <div className="char_block" title="#195;">#195;</div>
    </div>
  </div>
</div>
</div>
);
}
}
export default App;
{% endraw %}

```

[Functional-component]

APP.JSX

```
{% raw %}
```



```

/**
 * Rich Text Editor - Custom Tool Sample
 */
import { DialogComponent } from '@syncfusion/ej2-react-popups';
import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let rteObj;
  // set the value to Rich Text Editor
  let template = `<div style="display:block;"><p style="margin-
right:10px">The custom command "insert special character" is configured as
the last item of the toolbar. Click on the command and choose the special
character you want to include from the popup.</p></div>`;
  let selection = new NodeSelection();
  let ranges;
  let dialogObj;
  // Rich Text Editor items list
  let items = ['Bold', 'Italic', 'Underline', '|', 'Formats', 'Alignments',
'OrderedList',
  'UnorderedList', '|', 'CreateLink', 'Image', '|', 'SourceCode',
  {
    template: '<button class="e-tbar-btn e-btn" tabindex="-1"
id="custom_tbar" style="width:100%"><div class="e-tbar-btn-text"
style="font-weight: 500;"> &#937;</div></button>',
    undo: true,
    click: onClick.bind(this),
    tooltipText: 'Insert Symbol'
  }, '|', 'Undo', 'Redo'
];
  // Rich Text Editor ToolbarSettings
  let toolbarSettings = {
    items: items
  };
  let dlgButtons = [
    { buttonModel: { content: "Insert", isPrimary: true }, click:
onInsert.bind(this) },
    { buttonModel: { content: 'Cancel' }, click: onCancel }
  ];
  let header = 'Special Characters';
  let target = document.getElementById('rteSection');
  let height = 'auto';
  function dialogCreate() {
    const dialogCtn = document.getElementById('rteSpecial_char');
    dialogCtn.onclick = (e) => {
      const target = e.target;
      const activeEle = dialogObj.element.querySelector('.char_block.e-
active');
      if (target.classList.contains('char_block')) {
        target.classList.add('e-active');
        if (activeEle) {
          activeEle.classList.remove('e-active');
        }
      }
    };
  }
}

```

```

function onInsert() {
    const activeEle = dialogObj.element.querySelector('.char_block.e-
active');
    if (activeEle) {
ranges.insertNode(document.createTextNode(activeEle.textContent));
    }
    dialogOverlay();
}
function onClick() {
    ranges = selection.getRange(document);
    dialogObj.width = rteObj.element.offsetWidth * 0.5;
    dialogObj.content = document.getElementById('rteSpecial_char');
    dialogObj.dataBind();
    dialogObj.show();
}
function dialogOverlay() {
    const activeEle = dialogObj.element.querySelector('.char_block.e-
active');
    if (activeEle) {
        activeEle.classList.remove('e-active');
    }
    dialogObj.hide();
}
function onCancel(e) {
    const activeEle = this.element.querySelector('.char_block.e-active');
    if (activeEle) {
        activeEle.classList.remove('e-active');
    }
    this.hide();
}
const hideDiv = { display: "none" };
return (<div className='control-pane'>
    <div className='control-section e-rte-custom-tbar-section'
id="rteCustomTool">
        <div className='rte-control-section' id='rteSection'>
            <RichTextEditorComponent id="defaultRTE" ref={(scope) => { rteObj =
scope; }} valueTemplate={template} toolbarSettings={toolbarSettings}>
                <Inject services={[HtmlEditor, Toolbar, Link, Image,
QuickToolbar]}/>
            </RichTextEditorComponent>
            <DialogComponent id='customTbarDlg' ref={(scope) => { dialogObj =
scope; }} buttons={dlgButtons} overlayClick={dialogOverlay.bind(this)}
header={header} visible={false} showCloseIcon={false} target={'#rteSection'}
height={height} created={dialogCreate.bind(this)} isModal={true}
cssClass={'e-rte-elements'}/>
            <div id="customTbarDialog" style={hideDiv}>
                <div id="rteSpecial_char">
                    <div className="char_block" title="#94;">#94;</div>
                    <div className="char_block" title="#95;">#95;</div>
                    <div className="char_block" title="#96;">#96;</div>
                    <div className="char_block" title="#123;">#123;</div>
                    <div className="char_block" title="#124;">#124;</div>
                    <div className="char_block" title="#125;">#125;</div>
                    <div className="char_block" title="#126;">#126;</div>
                    <div className="char_block" title="#160;">#160;</div>
                    <div className="char_block" title="#161;">#161;</div>

```

```

        <div className="char_block" title="#162;">#162;</div>
        <div className="char_block" title="#163;">#163;</div>
        <div className="char_block" title="#164;">#164;</div>
        <div className="char_block" title="#165;">#165;</div>
        <div className="char_block" title="#x20B9;">#x20B9;</div>
        <div className="char_block" title="#166;">#166;</div>
        <div className="char_block" title="#167;">#167;</div>
        <div className="char_block" title="#168;">#168;</div>
        <div className="char_block" title="#169;">#169;</div>
        <div className="char_block" title="#170;">#170;</div>
        <div className="char_block" title="#171;">#171;</div>
        <div className="char_block" title="#172;">#172;</div>
        <div className="char_block" title="#173;">#173;</div>
        <div className="char_block" title="#174;">#174;</div>
        <div className="char_block" title="#175;">#175;</div>
        <div className="char_block" title="#176;">#176;</div>
        <div className="char_block" title="#177;">#177;</div>
        <div className="char_block" title="#178;">#178;</div>
        <div className="char_block" title="#179;">#179;</div>
        <div className="char_block" title="#180;">#180;</div>
        <div className="char_block" title="#181;">#181;</div>
        <div className="char_block" title="#182;">#182;</div>
        <div className="char_block" title="#183;">#183;</div>
        <div className="char_block" title="#184;">#184;</div>
        <div className="char_block" title="#185;">#185;</div>
        <div className="char_block" title="#186;">#186;</div>
        <div className="char_block" title="#187;">#187;</div>
        <div className="char_block" title="#188;">#188;</div>
        <div className="char_block" title="#189;">#189;</div>
        <div className="char_block" title="#190;">#190;</div>
        <div className="char_block" title="#191;">#191;</div>
        <div className="char_block" title="#192;">#192;</div>
        <div className="char_block" title="#193;">#193;</div>
        <div className="char_block" title="#194;">#194;</div>
        <div className="char_block" title="#195;">#195;</div>
    </div>
</div>
</div>
</div>
</div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Custom Tool Sample
 */
import { DialogComponent } from '@syncfusion/ej2-react-popups';
import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {

```

```

let rteObj: RichTextEditorComponent;
// set the value to Rich Text Editor
let template: string = `

Copyright © 2001 -2024 Syncfusion Inc.



1304


```

```

    }
    function onClick(): void {
        ranges = selection.getRange(document);
        dialogObj.width = (rteObj as any).element.offsetWidth * 0.5;
        dialogObj.content = document.getElementById('rteSpecial_char') as
HTMLInputElement;
        dialogObj.dataBind();
        dialogObj.show();
    }
    function dialogOverlay(): void {
        const activeEle: Element =
dialogObj.element.querySelector('.char_block.e-active') as HTMLInputElement;
        if (activeEle) {
            activeEle.classList.remove('e-active');
        }
        dialogObj.hide();
    }
    function onCancel(e: any): void {
        const activeEle: Element = (this as
any).element.querySelector('.char_block.e-active');
        if (activeEle) {
            activeEle.classList.remove('e-active');
        }
        (this as any).hide();
    }
    const hideDiv = { display: "none" };
    return (
        <div className='control-pane'>
            <div className='control-section e-rte-custom-tbar-section'
id="rteCustomTool">
                <div className='rte-control-section' id='rteSection'>
                    <RichTextEditorComponent id="defaultRTE" ref={(scope) => { rteObj =
scope! }}
                        valueTemplate={template} toolbarSettings={toolbarSettings}>
                        <Inject services={[HtmlEditor, Toolbar, Link, Image,
QuickToolbar]} />
                    </RichTextEditorComponent>
                    <DialogComponent id='customTbarDlg' ref={(scope) => { dialogObj =
scope! }}
                        buttons={dlgButtons} overlayClick={dialogOverlay.bind(this)}
header={header} visible={false}
                        showCloseIcon={false} target={'#rteSection'} height={height}
created={dialogCreate.bind(this)} isModal={true} cssClass={'e-rte-
elements'} />
                    <div id="customTbarDialog" style={hideDiv}>
                        <div id="rteSpecial_char">
                            <div className="char_block" title="#94;">#94;</div>
                            <div className="char_block" title="#95;">#95;</div>
                            <div className="char_block" title="#96;">#96;</div>
                            <div className="char_block" title="#123;">#123;</div>
                            <div className="char_block" title="#124;">#124;</div>
                            <div className="char_block" title="#125;">#125;</div>
                            <div className="char_block" title="#126;">#126;</div>
                            <div className="char_block" title="#160;">#160;</div>
                            <div className="char_block" title="#161;">#161;</div>
                            <div className="char_block" title="#162;">#162;</div>
                            <div className="char_block" title="#163;">#163;</div>

```

```

        <div className="char_block" title="#164;">#164;</div>
        <div className="char_block" title="#165;">#165;</div>
        <div className="char_block" title="#x20B9;">#x20B9;</div>
        <div className="char_block" title="#166;">#166;</div>
        <div className="char_block" title="#167;">#167;</div>
        <div className="char_block" title="#168;">#168;</div>
        <div className="char_block" title="#169;">#169;</div>
        <div className="char_block" title="#170;">#170;</div>
        <div className="char_block" title="#171;">#171;</div>
        <div className="char_block" title="#172;">#172;</div>
        <div className="char_block" title="#173;">#173;</div>
        <div className="char_block" title="#174;">#174;</div>
        <div className="char_block" title="#175;">#175;</div>
        <div className="char_block" title="#176;">#176;</div>
        <div className="char_block" title="#177;">#177;</div>
        <div className="char_block" title="#178;">#178;</div>
        <div className="char_block" title="#179;">#179;</div>
        <div className="char_block" title="#180;">#180;</div>
        <div className="char_block" title="#181;">#181;</div>
        <div className="char_block" title="#182;">#182;</div>
        <div className="char_block" title="#183;">#183;</div>
        <div className="char_block" title="#184;">#184;</div>
        <div className="char_block" title="#185;">#185;</div>
        <div className="char_block" title="#186;">#186;</div>
        <div className="char_block" title="#187;">#187;</div>
        <div className="char_block" title="#188;">#188;</div>
        <div className="char_block" title="#189;">#189;</div>
        <div className="char_block" title="#190;">#190;</div>
        <div className="char_block" title="#191;">#191;</div>
        <div className="char_block" title="#192;">#192;</div>
        <div className="char_block" title="#193;">#193;</div>
        <div className="char_block" title="#194;">#194;</div>
        <div className="char_block" title="#195;">#195;</div>
    </div>
  </div>
</div>
</div>
);
}
export default App;
{% enddraw %}

```

The focus will be lost while rendering the required component for the custom toolbar, causing it to render outside the Rich Text Editor and triggering a blur event. During that time, proper functionality will not be achievable. Therefore, it is recommended to set the cssClass property or class as `e-rte-elements` in the dependency component.

Quick inline toolbar

Quick commands are opened as context-menu on clicking the corresponding element. The commands must be passed as string collection to image, text, and link attributes of the [quickToolbarSettings](#) property.

| **Target element** | **Default quick toolbar items** |

| --- | --- |

| Image | 'Replace', 'Align', 'Caption', 'Remove', 'InsertLink', 'Display', 'AltText', and 'Dimension' |

| Link | 'Open', 'Edit', and 'UnLink' |

| Text | null
 (Any toolbar [items](#) in the Rich Text Editor can be configured here). |

| Table | 'TableHeader', 'TableRows', 'TableColumns', 'BackgroundColor', 'TableRemove', 'Alignments', 'TableCellVerticalAlign' and 'Styles' |

Custom tool can be added to the corresponding quick toolbar, using the [quickToolbarSettings](#) property.

The following sample demonstrates the option to insert the image to the Rich Text Editor content as well as option to rotate the image through the quick toolbar. The image rotation functionalities have been achieved through the [toolbarClick](#) event.

[Class-component]

APP.JSX

```
{% raw %}
/**
 * Rich Text Editor - Quick Inline Toolbar Sample
 */
import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  quickToolbarSettings = {
    image: [
      'Replace', 'Align', 'Caption', 'Remove', 'InsertLink',
      'OpenImageLink', '-',
      'EditImageLink', 'RemoveImageLink', 'Display', 'AltText',
      'Dimension',
      {
        template: '<button class="e-tbar-btn e-btn"
id="roatateLeft"><span class="e-btn-icon e-icons e-rotate-left"></span>',
        tooltipText: 'Rotate Left'
      },
      {
        template: '<button class="e-tbar-btn e-btn"
id="roatateRight"><span class="e-btn-icon e-icons e-rotate-right"></span>',
        tooltipText: 'Rotate Right'
      }
    ]
  };
  onToolbarClick(e) {
    const nodeObj = new NodeSelection();
    const range =
nodeObj.getRange(this.rteObj.contentModule.getDocument());
    const imgEle = nodeObj.getNodeCollection(range)[0];
    if (e.item.tooltipText === 'Rotate Right') {
      const transform = (imgEle.style.transform === '') ? 0 :
        parseInt(imgEle.style.transform.split('(')[1].split(' ')[0],
10);
      imgEle.style.transform = 'rotate(' + (transform + 90) + 'deg)';
    }
  }
}
```

```

    }
    else if (e.item.tooltipText === 'Rotate Left') {
      const transform = (imgEle.style.transform === '') ? 0 :
Math.abs(parseInt(imgEle.style.transform.split('(')[1].split(' ')[0], 10));
      imgEle.style.transform = 'rotate(-' + (transform + 90) + 'deg)';
    }
  }
  render() {
    return (<RichTextEditorComponent height={450} ref={(richtexteditor)
=> { this.rteObj = richtexteditor; }}
quickToolbarSettings={this.quickToolbarSettings}
toolbarClick={this.onToolbarClick} = this.onToolbarClick.bind(this)}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
  </RichTextEditorComponent>);
  }
}

```



```
export default App;
{% endraw %}
```

APP.TSX

```
{% raw %}
/**
 * Rich Text Editor - Quick Inline Toolbar Sample
 */
import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  public rteObj: RichTextEditorComponent;
  public quickToolbarSettings: object = {
    image: [
      'Replace', 'Align', 'Caption', 'Remove', 'InsertLink', 'OpenImageLink',
      '-',
      'EditImageLink', 'RemoveImageLink', 'Display', 'AltText', 'Dimension',
      {
        template: '<button class="e-tbar-btn e-btn" id="rotateLeft"><span
class="e-btn-icon e-icons e-rotate-left"></span>',
        tooltipText: 'Rotate Left'
      },
      {
        template: '<button class="e-tbar-btn e-btn" id="rotateRight"><span
class="e-btn-icon e-icons e-rotate-right"></span>',
        tooltipText: 'Rotate Right'
      }
    ]
  }
  public onToolbarClick(e: any): void {
    const nodeObj: NodeSelection = new NodeSelection();
    const range: Range = nodeObj.getRange((this.rteObj as
any).contentModule.getDocument());
    const imgEle: HTMLElement = nodeObj.getNodeCollection(range)[0] as
HTMLElement;
    if (e.item.tooltipText === 'Rotate Right') {
      const transform: number = (imgEle.style.transform === '') ? 0 :
parseInt((imgEle as any).style.transform.split('(')[1].split(' ')[0],
10);
      imgEle.style.transform = 'rotate(' + (transform + 90) + 'deg)';
    } else if (e.item.tooltipText === 'Rotate Left') {
      const transform: number = (imgEle.style.transform === '') ? 0 :
Math.abs(parseInt((imgEle as
any).style.transform.split('(')[1].split(' ')[0], 10));
      imgEle.style.transform = 'rotate(-' + (transform + 90) + 'deg)';
    }
  }
  public render() {
    return (
      <RichTextEditorComponent height={450} ref={(richtexteditor) => {
this.rteObj = richtexteditor! }}
quickToolbarSettings={this.quickToolbarSettings}
toolbarClick={this.onToolbarClick = this.onToolbarClick.bind(this)}>
```

```

    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
    you get") editor that provides the best user experience to create and update
    the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
  />
</RichTextEditorComponent>
);
}
}
export default App;
{% endraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - Quick Inline Toolbar Sample

```

```

*/
import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let rteObj;
  let quickToolbarSettings = {
    image: [
      'Replace', 'Align', 'Caption', 'Remove', 'InsertLink',
      'OpenImageLink', '-',
      'EditImageLink', 'RemoveImageLink', 'Display', 'AltText',
      'Dimension',
      {
        template: '<button class="e-tbar-btn e-btn"
id="rotateLeft"><span class="e-btn-icon e-icons e-rotate-left"></span>',
        tooltipText: 'Rotate Left'
      },
      {
        template: '<button class="e-tbar-btn e-btn"
id="rotateRight"><span class="e-btn-icon e-icons e-rotate-right"></span>',
        tooltipText: 'Rotate Right'
      }
    ]
  };
  function onToolbarClick(e) {
    const nodeObj = new NodeSelection();
    const range = nodeObj.getRange(rteObj.contentModule.getDocument());
    const imgEle = nodeObj.getNodeCollection(range)[0];
    if (e.item.tooltipText === 'Rotate Right') {
      const transform = (imgEle.style.transform === '') ? 0 :
        parseInt(imgEle.style.transform.split('(')[1].split(' ')[0],
10);
      imgEle.style.transform = 'rotate(' + (transform + 90) + 'deg)';
    }
    else if (e.item.tooltipText === 'Rotate Left') {
      const transform = (imgEle.style.transform === '') ? 0 :
        Math.abs(parseInt(imgEle.style.transform.split('(')[1].split(' ')[0], 10));
      imgEle.style.transform = 'rotate(-' + (transform + 90) + 'deg)';
    }
  }
  return (<RichTextEditorComponent height={450} ref={(richtexteditor) => {
rteObj = richtexteditor; }} quickToolbarSettings={quickToolbarSettings}
toolbarClick={onToolbarClick.bind(this)}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
    </ul>
  </RichTextEditorComponent>

```

```

        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
    </RichTextEditorComponent>;
  }
  export default App;
  {% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Quick Inline Toolbar Sample
 */
import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App () {
  let rteObj: RichTextEditorComponent;
  let quickToolbarSettings: object = {
    image: [
      'Replace', 'Align', 'Caption', 'Remove', 'InsertLink', 'OpenImageLink',
      '-',
      'EditImageLink', 'RemoveImageLink', 'Display', 'AltText', 'Dimension',
      {
        template: '<button class="e-tbar-btn e-btn" id="rootateLeft"><span
class="e-btn-icon e-icons e-rotate-left"></span>',
        tooltipText: 'Rotate Left'
      },
    ],
  },
  {

```

```

    template: '<button class="e-tbar-btn e-btn" id="rotateRight"><span
class="e-btn-icon e-icons e-rotate-right"></span>',
    tooltipText: 'Rotate Right'
  }
]
}
function onToolbarClick(e: any): void {
  const nodeObj: NodeSelection = new NodeSelection();
  const range: Range = nodeObj.getRange((rteObj as
any).contentModule.getDocument());
  const imgEle: HTMLElement = nodeObj.getNodeCollection(range)[0] as
HTMLElement;
  if (e.item.tooltipText === 'Rotate Right') {
    const transform: number = (imgEle.style.transform === '') ? 0 :
    parseInt((imgEle as any).style.transform.split('(')[1].split(' ')[0],
10);
    imgEle.style.transform = 'rotate(' + (transform + 90) + 'deg)';
  } else if (e.item.tooltipText === 'Rotate Left') {
    const transform: number = (imgEle.style.transform === '') ? 0 :
    Math.abs(parseInt((imgEle as
any).style.transform.split('(')[1].split(' ')[0], 10));
    imgEle.style.transform = 'rotate(-' + (transform + 90) + 'deg)';
  }
}
return (
  <RichTextEditorComponent height={450} ref={(richtexteditor) => { rteObj =
richtexteditor! }} quickToolbarSettings={quickToolbarSettings}
toolbarClick={onToolbarClick.bind(this)}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
    </ul>
  </RichTextEditorComponent>
);

```

```

        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}
export default App;
{% endraw %}

```

Rich Text Editor features are segregated into individual feature-wise modules. To use quick toolbar, inject the quick toolbar module using the `RichTextEditor.Inject(image, link)`.

Styling in React Rich text editor component

Font name and size

By default, the editor is initialized with font name as **Segoe UI** and font size as **10pt**. To change it, select a different font name and font size from the drop-down in the editor's toolbar.

To apply different font style for section of the content, select the text that you would like to change, and select a required font style from the drop-down to apply the changes to the selected text.

FontName DropDowns

The following table lists the default font name and width of the fontname drop-down and available list of font names.

Property	Default value
---	---
Default font name	'Segoe UI'
Width	65px
Items	{ text: 'Segoe UI', value: 'Segoe UI' }, { text: 'Arial', value: 'Arial,Helvetica,sans-serif' }, { text: 'Courier New', value: 'Courier New,Courier,monospace' }, { text: 'Georgia', value: 'Georgia,serif' }, { text: 'Impact', value: 'Impact,Charcoal,sans-serif' }, { text: 'Lucida Console', value: 'Lucida Console,Monaco,monospace' }, { text: 'Tahoma', value: 'Tahoma,Geneva,sans-serif' }, { text: 'Times New Roman', value: 'Times New Roman,Times,serif' }, { text: 'Trebuchet MS', value: 'Trebuchet MS,Helvetica,sans-serif' }, { text: 'Verdana', value: 'Verdana,Geneva,sans-serif' }

FontSize DropDowns

The following table list the default font size and width of the fontsize dropdown and available list of font size.

Property	Default value
---	---

| Default font size | 10 |

| Width | 35px |

| Items | { text: '8', value: '8pt' },
 { text: '10', value: '10pt' },
 { text: '12', value: '12pt' },
 { text: '14', value: '14pt' },
 { text: '18', value: '18pt' },
 { text: '24', value: '24pt' },
 { text: '36', value: '36pt' } |

The following sample demonstrates the option to add the font name and font size tools to the toolbar as well as modify the default width of the tools.

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - FontName and FontSize Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['FontName', 'FontSize']
  };
  fontFamily = {
    width: '60px'
  };
  fontSize = {
    width: '40px'
  };
  render() {
    return (
      <RichTextEditorComponent height={450}
      toolbarSettings={this.toolbarSettings} fontFamily={this.fontFamily}
      fontSize={this.fontSize}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
        </ul>
      </RichTextEditorComponent>
    );
  }
}
```

```

        </li>
        <li>
            <p>Supports third-party library integration.</p>
        </li>
        <li>
            <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
</RichTextEditorComponent>);
    }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - FontName and FontSize Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    private toolbarSettings: object = {
        items: ['FontName', 'FontSize']
    }
    private fontFamily: object = {
        width: '60px'
    }
    private fontSize: object = {
        width: '40px'
    }
    public render() {
        return (
            <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings} fontFamily={this.fontFamily}
fontSize={this.fontSize}>
                <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
                <p><b>Key features:</b></p>
                <ul>
                    <li>
                        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>

```



```

        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - FontName and FontSize Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['FontName', 'FontSize']
  };
  let fontFamily = {

```

```

        width: '60px'
    };
    let fontSize = {
        width: '40px'
    };
    return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings} fontFamily={fontFamily}
fontSize={fontSize}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
        <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
            <p>Capable of handling markdown editing.</p>
        </li>
        <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
            <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
            <p>Supports third-party library integration.</p>
        </li>
        <li>
            <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
</RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - FontName and FontSize Sample
 */

```

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings: object = {
    items: ['FontName', 'FontSize']
  }
  let fontFamily: object = {
    width: '60px'
  }
  let fontSize: object = {
    width: '40px'
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
fontFamily={fontFamily} fontSize={fontSize}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  )
}

```

```
);
}
export default App;
```

Custom font and size

Rich Text Editor supports to provide custom font and size with existing list.

If you want to add additional font names and font sizes to font drop-down, pass the font information as JSON data to the items field of the [fontSize](#) and the [fontFamily](#) property.

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - Custom FontName and FontSize Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['FontName', 'FontSize']
  };
  fontFamily = {
    items: [
      { text: 'Segoe UI', value: 'Segoe UI' },
      { text: 'Arial', value: 'Arial,Helvetica,sans-serif' },
      { text: 'Courier New', value: 'Courier New,Courier,monospace' },
      { text: 'Georgia', value: 'Georgia,serif' },
      { text: 'Impact', value: 'Impact,Charcoal,sans-serif' },
      { text: 'Calibri Light', value: 'CalibriLight' }
    ],
    width: '60px',
    default: 'Segoe UI'
  };
  fontSize = {
    items: [
      { text: '8', value: '8pt' },
      { text: '10', value: '10pt' },
      { text: '12', value: '12pt' },
      { text: '14', value: '14pt' },
      { text: '42', value: '42pt' }
    ],
    width: '40px',
    default: '10'
  };
  render() {
    return (
      <RichTextEditorComponent height={450}
        toolbarSettings={this.toolbarSettings} fontFamily={this.fontFamily}
        fontSize={this.fontSize}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
      </RichTextEditorComponent>
    );
  }
}
```

```

    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
  </RichTextEditorComponent>;
}
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Custom FontName and FontSize Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{},{}> {
  private toolbarSettings: object = {
    items: ['FontName', 'FontSize']
  }
  private fontFamily: object = {

```

```

    items: [
      { text: 'Segoe UI', value: 'Segoe UI' },
      { text: 'Arial', value: 'Arial,Helvetica,sans-serif' },
      { text: 'Courier New', value: 'Courier New,Courier,monospace' },
      { text: 'Georgia', value: 'Georgia,serif' },
      { text: 'Impact', value: 'Impact,Charcoal,sans-serif' },
      { text: 'Calibri Light', value: 'CalibriLight' }
    ],
    width: '60px',
    default: 'Segoe UI'
  }
  private fontSize: object = {
    items: [
      { text: '8', value: '8pt' },
      { text: '10', value: '10pt' },
      { text: '12', value: '12pt' },
      { text: '14', value: '14pt' },
      { text: '42', value: '42pt' }
    ],
    width: '40px',
    default: '10'
  }
  public render() {
    return (
      <RichTextEditorComponent height={450}
        toolbarSettings={this.toolbarSettings} fontFamily={this.fontFamily}
        fontSize={this.fontSize}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
        you get") editor that provides the best user experience to create and update
        the content. Users can format their content using standard toolbar
        commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
            on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
            developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
        </ul>
      </RichTextEditorComponent>
    )
  }
}

```

```

        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
    </li>
    <li>
        <p>Contains undo/redo manager.</p>
    </li>
    <li>
        <p>Creates bulleted and numbered lists.</p>
    </li>
</ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Custom FontName and FontSize Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    let toolbarSettings = {
        items: ['FontName', 'FontSize']
    };
    let fontFamily = {
        items: [
            { text: 'Segoe UI', value: 'Segoe UI' },
            { text: 'Arial', value: 'Arial,Helvetica,sans-serif' },
            { text: 'Courier New', value: 'Courier New,Courier,monospace' },
            { text: 'Georgia', value: 'Georgia,serif' },
            { text: 'Impact', value: 'Impact,Charcoal,sans-serif' },
            { text: 'Calibri Light', value: 'CalibriLight' }
        ],
        width: '60px'
    };
    let fontSize = {
        items: [
            { text: '8', value: '8pt' },
            { text: '10', value: '10pt' },
            { text: '12', value: '12pt' },
            { text: '14', value: '14pt' },
            { text: '42', value: '42pt' }
        ],
        width: '40px'
    };
}

```

```

    return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings} fontFamily={fontFamily}
fontSize={fontSize}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
        <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
            <p>Capable of handling markdown editing.</p>
        </li>
        <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
            <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
            <p>Supports third-party library integration.</p>
        </li>
        <li>
            <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
</RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Custom FontName and FontSize Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {

```



```

let toolbarSettings: object = {
  items: ['FontName', 'FontSize']
}
let fontFamily: object = {
  items: [
    { text: 'Segoe UI', value: 'Segoe UI' },
    { text: 'Arial', value: 'Arial,Helvetica,sans-serif' },
    { text: 'Courier New', value: 'Courier New,Courier,monospace' },
    { text: 'Georgia', value: 'Georgia,serif' },
    { text: 'Impact', value: 'Impact,Charcoal,sans-serif' },
    { text: 'Calibri Light', value: 'CalibriLight' }
  ],
  width: '60px'
}
let fontSize: object = {
  items: [
    { text: '8', value: '8pt' },
    { text: '10', value: '10pt' },
    { text: '12', value: '12pt' },
    { text: '14', value: '14pt' },
    { text: '42', value: '42pt' }
  ],
  width: '40px'
}
return (
  <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
fontFamily={fontFamily} fontSize={fontSize}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
    </ul>
  </RichTextEditorComponent>
)

```

```

        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}
export default App;

```

Font and Background color

To apply font color or background color for a selected content of RTE, use font color and background color tools.

Rich Text Editor supports to provide customs font color and background color with existing list through the [colorCode](#) field of `fontColor` and `backgroundColor`.

The `FontColor` and the `BackgroundColor` property has two [mode](#) Picker and Palette. Palette mode has predefined set of `colorCode` and in the picker mode, more colors has been provided. Through [modeSwitcher](#), you can able to switch between these two options.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - Font and Background Color Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['FontColor', 'BackgroundColor']
  };
  fontColor = {
    modeSwitcher: true
  };
  backgroundColor = {
    modeSwitcher: true
  };
  render() {
    return (
      <RichTextEditorComponent height={450}
      toolbarSettings={this.toolbarSettings} fontColor={this.fontColor}
      backgroundColor={this.backgroundColor}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>

```

```

        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
    </RichTextEditorComponent>;
  }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Font and Background Color Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private toolbarSettings: object = {
    items: ['FontColor', 'BackgroundColor']
  }
  private fontColor: object = {
    modeSwitcher : true
  }
}

```

```

private backgroundColor: object = {
  modeSwitcher : true
}
public render() {
  return (
    <RichTextEditorComponent height={450}
    toolbarSettings={this.toolbarSettings} fontColor={this.fontColor}
    backgroundColor={this.backgroundColor}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]**APP.JSX**

```

/**
 * Rich Text Editor - Font and Background Color Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['FontColor', 'BackgroundColor']
  };
  let fontColor = {
    modeSwitcher: true
  };
  let backgroundColor = {
    modeSwitcher: true
  };
  return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings} fontColor={fontColor}
backgroundColor={backgroundColor}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
    </ul>
  </RichTextEditorComponent>

```

```

        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
    </RichTextEditorComponent>);
  }
  export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Font and Background Color Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings: object = {
    items: ['FontColor', 'BackgroundColor']
  }
  let fontColor: object = {
    modeSwitcher : true
  }
  let backgroundColor: object = {
    modeSwitcher : true
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
fontColor={fontColor} backgroundColor={backgroundColor}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>

```

```

        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}
export default App;

```

Editor content styles

By default, The content styles of Rich Text Editor are not returned while retrieving HTML value from the editor. So, the styles are not applied when using the HTML value outside of the editor. To get the styles to Rich Text Editor's content for your application, You can copy and use the below styles directly in your application. The styles listed below which used in the UI elements of the Rich Text Editor.

Make sure to add a CSS class 'e-rte-content' to the content container.

```

`css
.e-rte-content p {
margin: 0 0 10px;
margin-bottom: 10px;
}
.e-rte-content li {
margin-bottom: 10px;
}
.e-rte-content h1 {
font-size: 2.17em;
font-weight: 400;
line-height: 1;
margin: 10px 0;
}
.e-rte-content h2 {
font-size: 1.74em;

```

```
font-weight: 400;
margin: 10px 0;
}
.e-rte-content h3 {
font-size: 1.31em;
font-weight: 400;
margin: 10px 0;
}
.e-rte-content h4 {
font-size: 1em;
font-weight: 400;
margin: 0;
}
.e-rte-content h5 {
font-size: 0.8em;
font-weight: 400;
margin: 0;
}
.e-rte-content h6 {
font-size: 0.65em;
font-weight: 400;
margin: 0;
}
.e-rte-content blockquote {
margin: 10px 0;
margin-left: 0;
padding-left: 5px;
}
.e-rte-content pre {
background-color: inherit;
border: 0;
border-radius: 0;
color: #333;
```



```
font-size: inherit;
line-height: inherit;
margin: 0 0 10px;
overflow: visible;
padding: 0;
white-space: pre-wrap;
word-break: inherit;
word-wrap: break-word;
}
.e-rte-content strong, .e-rte-content b {
font-weight: 700;
}
.e-rte-content a {
text-decoration: none;
-webkit-user-select: auto;
-ms-user-select: auto;
user-select: auto;
}
.e-rte-content a:hover {
text-decoration: underline;
}
.e-rte-content h3 + h4,
.e-rte-content h4 + h5,
.e-rte-content h5 + h6 {
margin-top: 00.6em;
}
.e-rte-content .e-rte-image.e-imgbreak {
border: 0;
cursor: pointer;
display: block;
float: none;
margin: 5px auto;
max-width: 100%;
```

```
position: relative;
}
.e-rte-content .e-rte-image {
border: 0;
cursor: pointer;
display: block;
float: none;
margin: auto;
max-width: 100%;
position: relative;
}
.e-rte-content .e-rte-image.e-imginline {
display: inline-block;
float: none;
margin-left: 5px;
margin-right: 5px;
max-width: calc(100% - (2 * 5px));
vertical-align: bottom;
}
.e-rte-content .e-rte-image.e-imgcenter {
cursor: pointer;
display: block;
float: none;
margin: 5px auto;
max-width: 100%;
position: relative;
}
.e-rte-content .e-rte-image.e-imgleft {
float: left;
margin: 0 5px 0 0;
text-align: left;
}
.e-rte-content .e-rte-image.e-imgright {
```

```
float: right;
margin: 0 0 0 5px;
text-align: right;
}

.e-rte-content .e-rte-img-caption {
display: inline-block;
margin: 5px auto;
max-width: 100%;
position: relative;
}

.e-rte-content .e-rte-img-caption.e-caption-inline {
display: inline-block;
margin: 5px auto;
margin-left: 5px;
margin-right: 5px;
max-width: calc(100% - (2 * 5px));
position: relative;
text-align: center;
vertical-align: bottom;
}

.e-rte-content .e-rte-img-caption.e-imgcenter {
display: block;
}

.e-rte-content .e-rte-img-caption .e-rte-image.e-imgright,
.e-rte-content .e-rte-img-caption .e-rte-image.e-ingleft {
float: none;
margin: 0;
}

.e-rte-content .e-rte-table {
border-collapse: collapse;
empty-cells: show;
}

.e-rte-content .e-rte-table td,
```

```
.e-rte-content .e-rte-table th {  
border: 1px solid #bdbdbd;  
height: 20px;  
min-width: 20px;  
padding: 2px 5px;  
vertical-align: middle;  
}  
.e-rte-content .e-rte-table.e-dashed-border td,  
.e-rte-content .e-rte-table.e-dashed-border th {  
border-style: dashed;  
}  
.e-rte-content .e-rte-img-caption .e-img-inner {  
box-sizing: border-box;  
display: block;  
font-size: 16px;  
font-weight: initial;  
margin: auto;  
opacity: .9;  
position: relative;  
text-align: center;  
width: 100%;  
}  
.e-rte-content .e-rte-img-caption .e-img-wrap {  
display: inline-block;  
margin: auto;  
padding: 0;  
width: 100%;  
}  
.e-rte-content blockquote {  
border-left: solid 2px #333;  
}  
.e-rte-content a {  
color: #2e2ef1;
```

```
}
.e-rte-content .e-rte-table th {
background-color: #e0e0e0;
}
`
```

Image in React Rich text editor component

Rich Text Editor allows to insert images from online sources as well as local computer where you want to insert the image in your content. For inserting the image to the Rich Text Editor, the following list of options have been provided in the [insertImageSettings](#).

| Options | Description |

| --- | --- |

| allowedTypes | Specifies the extensions of the image types allowed to insert on bowering and passing the extensions with comma separators.

 For example, pass allowedTypes as .jpg and .png. |

| display | Sets the default display for an image when it is inserted into the Rich Text Editor.

 Possible options are: 'inline' and 'block'. |

| width | Sets the default width of the image when it is inserted in the Rich Text Editor. |

| height | Sets the default height of the image when it is inserted in the Rich Text Editor. |

| saveUrl | Provides URL to map the action result method to save the image. |

| path | Specifies the location to store the image. |

Rich Text Editor features are segregated into individual feature-wise modules. To use image and link tool, inject image module using the `RichTextEditor.Inject(image)`.

Upload options

Through the 'browse' option, select the image from the local machine and insert into the Rich Text Editor content.

If the path field is not specified in the [insertImageSettings](#), the image will be transferred into base 64 and blob url for the image will be created and the generated url will be set to the src property of img tag.

```
`javascript
```

```

```

```
`
```

If you want to insert a lot of tiny images in the editor and don't want a specific physical location for saving images, you can opt to save format as Base64.

In the following sample, the image has been loaded from the local machine and it will be saved in the given location.

[Class-component]

APP.JSX

```
/**
```

```

* Rich Text Editor - Image Sample
*/
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['Image']
  };
  render() {
    return (<RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
    </RichTextEditorComponent>);
  }
}

```

```
export default App;
```

APP.TSX

```
/**
 * Rich Text Editor - Image Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private toolbarSettings: object = {
    items: ['Image']
  }
  public render() {
    return (
      <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
          </li>
          <li>
            <p>Contains undo/redo manager.</p>
          </li>
          <li>

```

```

        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
  />
</RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Image Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['Image']
  };
  return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>

```



```

        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
</RichTextEditorComponent>;
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Image Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
    let toolbarSettings: object = {
        items: ['Image']
    }
    return (
        <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}>
            <p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides the best user experience to create and update the content.
                Users can format their content using standard toolbar commands.</p>
            <p><b>Key features:</b></p>
            <ul>
                <li>
                    <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
                </li>
                <li>
                    <p>Capable of handling markdown editing.</p>
                </li>
                <li>
                    <p>Contains a modular library to load the necessary functionality on demand.</p>
                </li>
                <li>
                    <p>Provides a fully customizable toolbar.</p>
                </li>
                <li>
                    <p>Provides HTML view to edit the source directly for developers.</p>
                </li>
                <li>
                    <p>Supports third-party library integration.</p>
                </li>
            </ul>
        </RichTextEditorComponent>
    );
}

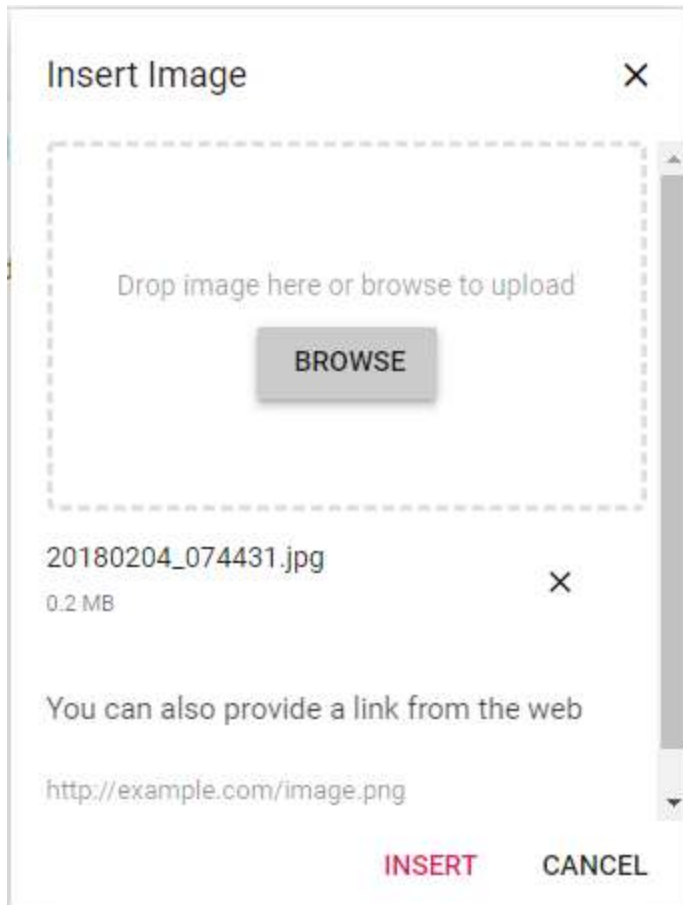
```

```
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}
export default App;
```

Image delete

To remove an image from the Rich Text Editor content, select the image and click **Remove** tool from the quick toolbar. It will delete the image from the RTE content as well as from the service location if the `saveUrl` is given.

Once you select the image from the local machine, the URL for the image will be generate. From there, you can remove the image from the service location by clicking the cross icon.



The following sample explains, how to configure `removeUrl` to remove a saved image from the remote service location, when the following image remove actions are performed:

- `delete` key action.
- `backspace` key action.
- Removing uploaded image file from the insert image dialog.
- Deleting image using the quick toolbar `remove` option.

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - RemoveUrl Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['Image']
  };
  insertImageSettings = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
```

```

        removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
    };
    render() {
        return (<RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}
insertImageSettings={insertImageSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
                <p>Capable of handling markdown editing.</p>
            </li>
            <li>
                <p>Contains a modular library to load the necessary functionality
on demand.</p>
            </li>
            <li>
                <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
                <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
                <p>Supports third-party library integration.</p>
            </li>
            <li>
                <p>Allows preview of modified content before saving it.</p>
            </li>
            <li>
                <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
            </li>
            <li>
                <p>Contains undo/redo manager.</p>
            </li>
            <li>
                <p>Creates bulleted and numbered lists.</p>
            </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
    </RichTextEditorComponent>);
    }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - RemoveUrl Sample

```

```

*/
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}> {
  private toolbarSettings: object = {
    items: ['Image']
  }
  private insertImageSettings: object = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
    removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
  }
  public render() {
    return (
      <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}
insertImageSettings={this.insertImageSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
          </li>
          <li>
            <p>Contains undo/redo manager.</p>
          </li>
          <li>
            <p>Creates bulleted and numbered lists.</p>
          </li>
        </ul>
      </RichTextEditorComponent>
    );
  }
}

```

```

        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
    />
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - RemoveUrl Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['Image']
  };
  let insertImageSettings = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
    removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
  };
  return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings} insertImageSettings={insertImageSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
    </ul>

```

```

    <li>
      <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
      <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
    </li>
    <li>
      <p>Contains undo/redo manager.</p>
    </li>
    <li>
      <p>Creates bulleted and numbered lists.</p>
    </li>
  </ul>
  <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
</RichTextEditorComponent>;
}
export default App;

```

APP.TSX

```

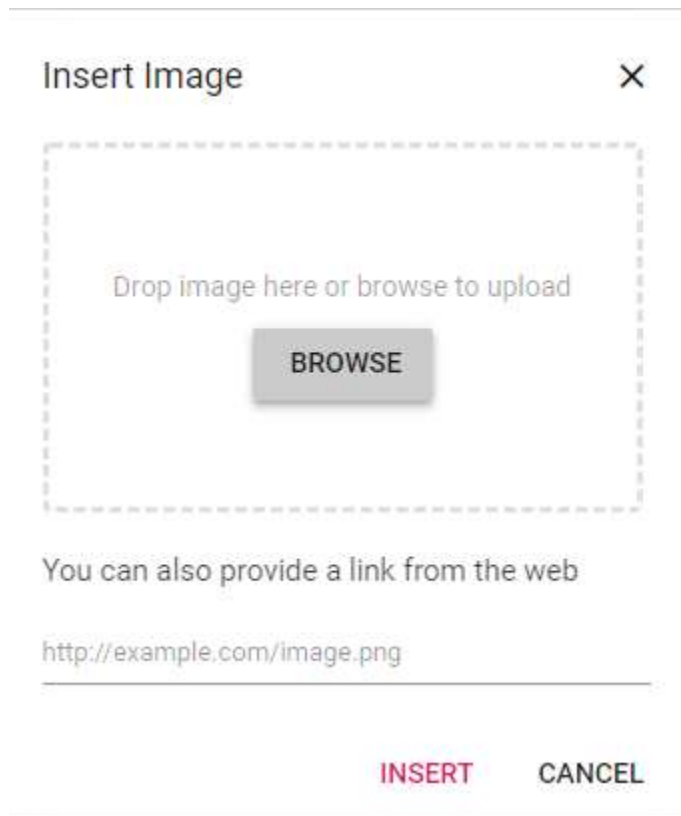
/**
 * Rich Text Editor - RemoveUrl Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings: object = {
    items: ['Image']
  }
  let insertImageSettings: object = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
    removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
insertImageSettings={insertImageSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>

```

```
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}
export default App;
```

Insert from web

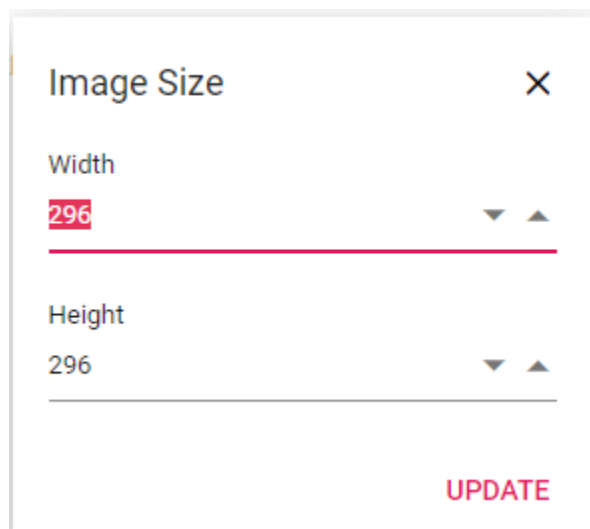
To insert an image from the online source like Google, Ping, etc., you should enable the image tool on the editor's toolbar. By default, the image tool opens a simple dialog which allows you to insert an image from online source.



Dimension

Sets the default width and height of the image when it is inserted in the Rich Text Editor using [width](#) and [height](#) of the `insertImageSettings`.

Through the quick toolbar, change the width and height using **Change Size** option. Once you click, the Image Size dialog box will open as follows. In that, you can specify the width and height of the image in pixel.



Caption and Alt Text

Image caption and alternative text can be specified for the inserted image in the Rich Text Editor through the quick toolbar options such as, Image Caption and Alternative Text.

Through the Alternative Text option, set the alternative text for the image, when the image is not upload successfully into the Rich Text Editor.

By clicking the Image Caption, the image will get wrapped in an image element with a caption. Then, you can type caption content inside the Rich Text Editor.

Display position

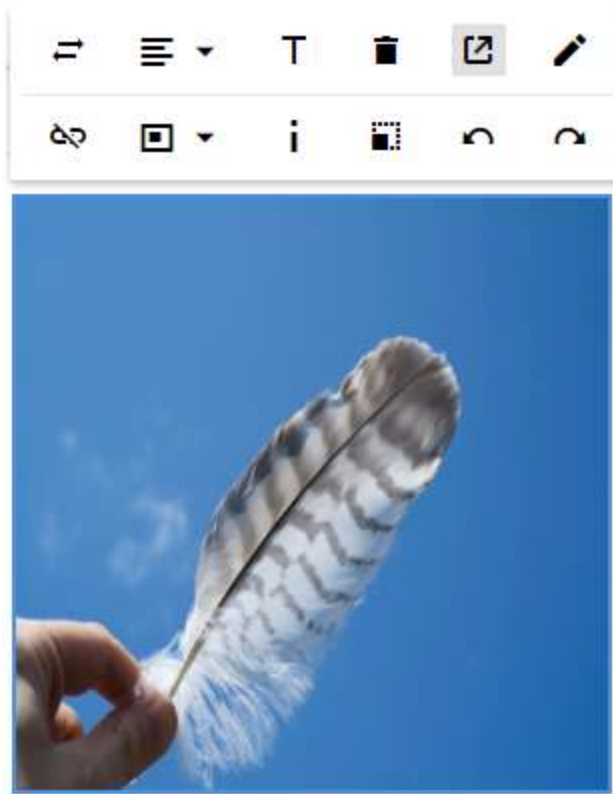
Sets the default display for an image when it is inserted in the Rich Text Editor using [display](#) field in [insertImageSettings](#). It has two possible options: 'inline' and 'block'.

`ts

```
let defaultRTE: RichTextEditor = new RichTextEditor({  
  insertImageSettings: {  
    display: 'inline'  
  }  
});  
defaultRTE.appendTo('#defaultRTE');  
`
```

Image with link

The hyperlink itself can be an image in Rich Text Editor. If the image given as hyperlink, the remove, edit, and open link will be added to the quick toolbar of image. For further details about link, see the [link](#) documentation.



Resize

Rich Text Editor has a built-in image inserting support. The resize points will be appearing on each corner of image when focus. So, users can resize the image using mouse points or thumb through the resize points easily. Also, the resize calculation will be done based on aspect ratio.



Drag and Drop

By default, the Rich Text Editor allows you to insert images by drag-and-drop from the local file system such as Windows Explorer into the content editor area. And, you can upload the images to the server before inserting into the editor by configuring the `saveUrl` property. The images can be repositioned anywhere within the editor area by dragging and dropping the image.

In the following sample, you can see feature demo.

[Class-component]

APP.JSX

```

/**
 * Initilaize Rich Text Editor from React element
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
    insertImageSettings = {
        saveUrl:
'https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save'
    };
    render() {
        return (<RichTextEditorComponent
insertImageSettings={this.insertImageSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is
what you get") editor that provides the best user experience to create and
update the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62;
modes</p>
            </li>
            <li>
                <p>Capable of handling markdown editing.</p>
            </li>
            <li>
                <p>Contains a modular library to load the necessary
functionality on demand.</p>
            </li>
            <li>
                <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
                <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
                <p>Supports third-party library integration.</p>
            </li>
            <li>
                <p>Allows preview of modified content before saving
it.</p>
            </li>
            <li>
                <p>Handles images, hyperlinks, video, hyperlinks,
uploads, etc.</p>
            </li>
            <li>
                <p>Contains undo/redo manager.</p>
            </li>
            <li>
                <p>Creates bulleted and numbered lists.</p>

```

```

        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor,
QuickToolbar]}/>
    </RichTextEditorComponent>;
  }
}
export default App;

```

APP.TSX

```

/**
 * Initilaize Rich Text Editor from React element
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  public insertImageSettings: object = {
    saveUrl :
'https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save'
  };
  public render() {
    return (
      <RichTextEditorComponent
insertImageSettings={this.insertImageSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is
what you get") editor that provides the best user experience to create and
update the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62;
modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary
functionality on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>

```

```

        <p>Allows preview of modified content before saving
it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks,
uploads, etc.</p>
        </li>
        <li>
            <p>Contains undo/redon manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor,
QuickToolbar]} />
</RichTextEditorComponent>
    );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Initilaize Rich Text Editor from React element
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    let insertImageSettings = {
        saveUrl:
'https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save'
    };
    return (<RichTextEditorComponent
insertImageSettings={insertImageSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
                <p>Capable of handling markdown editing.</p>
            </li>
            <li>
                <p>Contains a modular library to load the necessary
functionality on demand.</p>
            </li>

```

```

        <li>
            <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
            <p>Supports third-party library integration.</p>
        </li>
        <li>
            <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
</RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Initilaize Rich Text Editor from React element
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    let insertImageSettings: object = {
        saveUrl :
'https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save'
    };
    return (
        <RichTextEditorComponent insertImageSettings={insertImageSettings}>
            <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
            <p><b>Key features:</b></p>
            <ul>
                <li>
                    <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
                </li>
                <li>
                    <p>Capable of handling markdown editing.</p>
                </li>
            </ul>
        </RichTextEditorComponent>
    );
}

```

```

        </li>
        <li>
          <p>Contains a modular library to load the necessary
functionality on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
export default App;

```

Drag and drop with specific extension images

You can allow the specific images alone to be uploaded using the the `allowedTypes` property. By default, the Rich Text Editor allows the JPG, JPEG, and PNG formats. You can configure this formats as follows.

```

`ts
insertImageSettings: {
  allowedTypes: ['.jpg']
}
`

```

Prevent drag and drop action

You can prevent drag-and-drop action by setting the `actionBegin` argument cancel value to true. The following code shows how to prevent the drag-and-drop.

```

`ts
actionBegin: function (args: any): void {

```



```

if(args.type === 'drop' || args.type === 'dragstart') {
  args.cancel = true;
}
}
`

```

Audio in React Rich text editor component

The Rich Text Editor allows you to insert audio from online sources and local computers and then insert them into your content. You can insert the audio with the following list of options in the [insertAudioSettings](#) property.

Options	Description
-----	-----
allowedTypes	Specifies the extensions of the audio types allowed to insert on bowering and passing the extensions with comma separators. For example, pass allowedTypes as <code>.mp3</code> , <code>.wav</code> , <code>.m4a</code> and <code>.wma</code> .
layoutOption	Sets the default display for audio when it is inserted into the Rich Text Editor. Possible options are <code>Inline</code> and <code>Break</code> .
saveFormat	Sets the default save format of the audio element when inserted. Possible options are: <code>Blob</code> and <code>Base64</code> .
saveUrl	Provides URL to map the action result method to save the audio.
removeUrl	Provides URL to map the action result method to remove the audio.
path	Specifies the location to store the audio.

Configure audio tool in the toolbar

You can add an `audio` tool in the Rich Text Editor toolbar using the `toolbarSettings.items` property.

Rich Text Editor features are segregated into individual feature-wise modules. To use audio, inject the `Audio` module in `services`.

To configure the `Audio` toolbar item, refer to the below code.

[Class-component]

APP.JSX

```

import * as React from 'react';
import { HtmlEditor, Audio, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
class App extends React.Component {
  toolbarSettings = {
    items: ['Audio']
  };
  insertAudioSettings = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
    removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
  };
}

```

```

render() {
    return (<RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}
insertAudioSettings={this.insertAudioSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
        <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
            <p>Capable of handling markdown editing.</p>
        </li>
        <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
            <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
            <p>Supports third-party library integration.</p>
        </li>
        <li>
            <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]}>/>
</RichTextEditorComponent>);
}
}
export default App;

```

APP.TSX

```

import * as React from 'react';
import { HtmlEditor, Audio, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';

```

```

class App extends React.Component<{},{}> {
  private toolbarSettings: object = {
    items: ['Audio']
  }
  private insertAudioSettings: object = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
    removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
  }
  public render() {
    return (
      <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings} insertAudioSettings =
{this.insertAudioSettings} >
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
          </li>
          <li>
            <p>Contains undo/redo manager.</p>
          </li>
          <li>
            <p>Creates bulleted and numbered lists.</p>
          </li>
        </ul>
        <Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]}
/>
      </RichTextEditorComponent>
    );
  }
}

```

```

    }
  }
  export default App;

```

[Functional-component]

APP.JSX

```

import * as React from 'react';
import { HtmlEditor, Audio, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
function App() {
  let toolbarSettings = {
    items: ['Audio']
  };
  let insertAudioSettings = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
    removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
  };
  return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings} insertAudioSettings={insertAudioSettings}>
  <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
  Users can format their content using standard toolbar commands.</p>
  <p><b>Key features:</b></p>
  <ul>
    <li>
      <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
    </li>
    <li>
      <p>Capable of handling markdown editing.</p>
    </li>
    <li>
      <p>Contains a modular library to load the necessary functionality
on demand.</p>
    </li>
    <li>
      <p>Provides a fully customizable toolbar.</p>
    </li>
    <li>
      <p>Provides HTML view to edit the source directly for
developers.</p>
    </li>
    <li>
      <p>Supports third-party library integration.</p>
    </li>
    <li>
      <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
      <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
    </li>
  </ul>

```

```

        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]}/>
  </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

import * as React from 'react';
import { HtmlEditor, Audio, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
function App() {
  let toolbarSettings: object = {
    items: ['Audio']
  }
  let insertAudioSettings: object = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
    removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
insertAudioSettings = {insertAudioSettings} >
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>

```

```

        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
export default App;

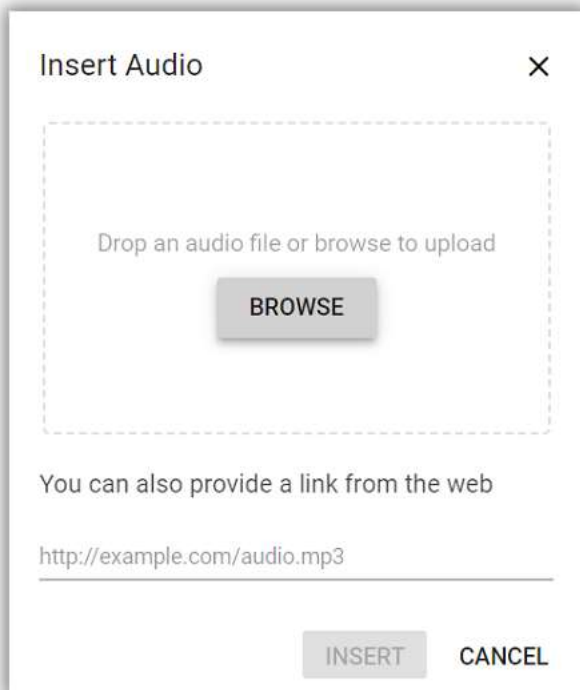
```

Insert audio from the web

You can insert audio from either the hosted link or the local machine, by clicking the audio button in the editor's toolbar. On clicking the audio button, a dialog opens, which allows you to insert audio from the web URL.

Insert from web URL

By default, the audio tool opens the audio dialog, allowing you to insert audio from an online source. Inserting the URL will be added to the `src` attribute of the `<source>` tag.



Insert audio from local machine

You can use the `browse` option on the audio dialog, to select the audio from the local machine and insert it into the Rich Text Editor content.

If the path field is not specified in the [insertAudioSettings](#), the audio will be converted into the `Blob` URL or `Base64` and inserted inside the Rich Text Editor.

Restrict audio upload based on size

You can restrict the audio uploaded from the local machine when the uploaded audio file size is greater than the allowed size by using the [fileUploading](#) event.

The file size in the argument will be returned in `bytes`.

In the following illustration, the audio size has been validated before uploading, and it is determined whether the audio has been uploaded or not.

[Class-component]

```

`ts

import { HtmlEditor, Audio, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';

import * as React from 'react';

import { UploadingEventArgs } from '@syncfusion/ej2-inputs';

class App extends React.Component<{},{}> {
  private toolbarSettings: object = {
    items: ['Audio']
  }

  private insertAudioSettings: object = {
    saveUrl: "https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save",
    path: "../Files/"
  }

  private onFileUpload (args: UploadingEventArgs): void {
    alert("RTE")

    let sizeInBytes: number = args.fileData.size;
    let fileSize: number = 500000;

    if (fileSize < sizeInBytes) {
      args.cancel = true;
    }
  }

  public render() {
    return (

```

```

<RichTextEditorComponent height={450} toolbarSettings={this.toolbarSettings} insertAudioSettings={
this.insertAudioSettings} fileUploading={this.onFileUpload}>
<Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
}
export default App;
`

```

[Functional-component]

```

`ts
import { HtmlEditor, Audio, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
import { UploadingEventArgs } from '@syncfusion/ej2-inputs';
function App() {
let toolbarSettings: object = {
items: ['Audio']
}
let insertAudioSettings: object = {
saveUrl: "https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save",
path: "../Files/"
}
function onFileUpload (args: UploadingEventArgs): void {
alert("RTE")
let sizeInBytes: number = args.fileData.size;
let fileSize: number = 500000;
if (fileSize < sizeInBytes) {
args.cancel = true;
}
}
return (
<RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
insertAudioSettings={insertAudioSettings} fileUploading={onFileUpload}>

```



```

<Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
export default App;
`

```

Server-side action

The selected audio can be uploaded to the required destination using the controller action below. Map this method name in [insertAudioSettings.saveUrl](#) and provide the required destination path through [insertAudioSettings.path](#) properties.

If you want to insert lower-sized audio files in the editor and don't want a specific physical location for saving the audio, you can opt to save the format as **Base64**.

In the following code blocks, the audio module has been injected and can insert the audio files saved in the specified path.

[Class-component]

```

`ts
import { HtmlEditor, Audio, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component<{},{}> {
  private toolbarSettings: object = {
    items: ['Audio']
  }
  private insertAudioSettings: object = {
    saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
    path: "[SERVICEHOSTEDPATH]/Files/"
  }
  public render() {
    return (
      <RichTextEditorComponent height={450} toolbarSettings={this.toolbarSettings}
        insertAudioSettings={this.insertAudioSettings}>
        <Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]} />
      </RichTextEditorComponent>
    );
  }
}

```

```

}
export default App;
`
[Functional-component]
`ts
import { HtmlEditor, Audio, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App(){
let toolbarSettings: object = {
items: ['Audio']
}
let insertAudioSettings: object = {
saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
path: "[SERVICEHOSTEDPATH]/Files/"
}
return (
<RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
insertAudioSettings={insertAudioSettings}>
<Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
export default App;
`
`csharp
using System;
using System.IO;
using FileUpload.Models;
using System.Diagnostics;
using System.Net.Http.Headers;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Http;
using System.Collections.Generic;

```

```
using Microsoft.AspNetCore.Hosting;
namespace FileUpload.Controllers
{
    public class HomeController : Controller
    {
        private IHostingEnvironment hostingEnv;
        public HomeController(IHostingEnvironment env)
        {
            hostingEnv = env;
        }
        public IActionResult Index()
        {
            return View();
        }
        [AcceptVerbs("Post")]
        public void SaveFiles(IList<IFormFile> UploadFiles)
        {
            try
            {
                foreach (IFormFile file in UploadFiles)
                {
                    if (UploadFiles != null)
                    {
                        string filename = ContentDispositionHeaderValue.Parse(file.ContentDisposition).FileName.Trim("");
                        filename = hostingEnv.WebRootPath + "\\Files" + $"@\"{filename}";
                        // Create a new directory, if it does not exists
                        if (!Directory.Exists(hostingEnv.WebRootPath + "\\Files"))
                        {
                            Directory.CreateDirectory(hostingEnv.WebRootPath + "\\Files");
                        }
                        if (!System.IO.File.Exists(filename))
                        {
                            using (FileStream fs = System.IO.File.Create(filename))
```

```

{
file.CopyTo(fs);
fs.Flush();
}
Response.StatusCode = 200;
}
}
}
}
catch (Exception)
{
Response.StatusCode = 204;
}
}
[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error()
{
return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
}
}
}
,

```

Audio save format

The audio files can be saved as **Blob** or **Base64** url by using the [insertAudioSettings.saveFormat](#) property, which is of enum type and the generated url will be set to the **src** attribute of the **<source>** tag.

By default, the files are saved in the **Blob** format.

```
`ts
```

```
<audio>
```

```
<source src="blob:http://ej2.syncfusion.com/3ab56a6e-ec0d-490f-85a5-f0aeb0ad8879"
type="audio/mp3" >
```

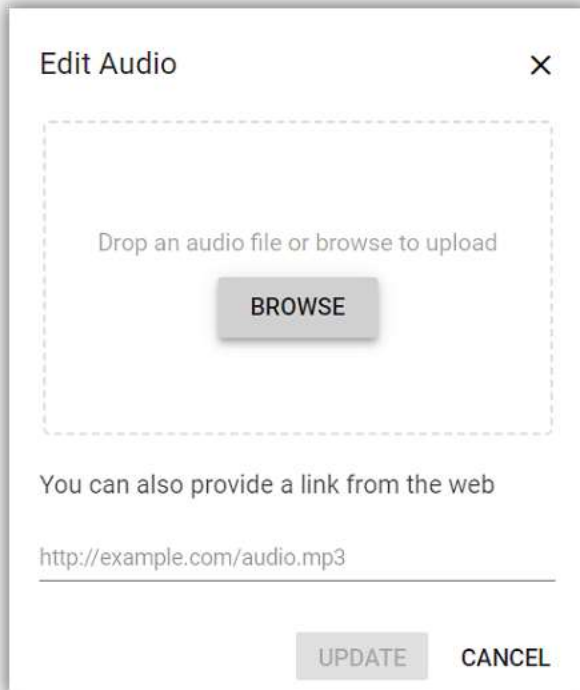
```
</audio>
```

```
<audio>
```

```
<source src="data:audio/mp3;base64,iVBORw0KGgoAAAANSUhEUgAAADAAAAAwCAYAAABXAvmHA"
type="audio/mp3" >
</audio>
`
```

Replacing audio

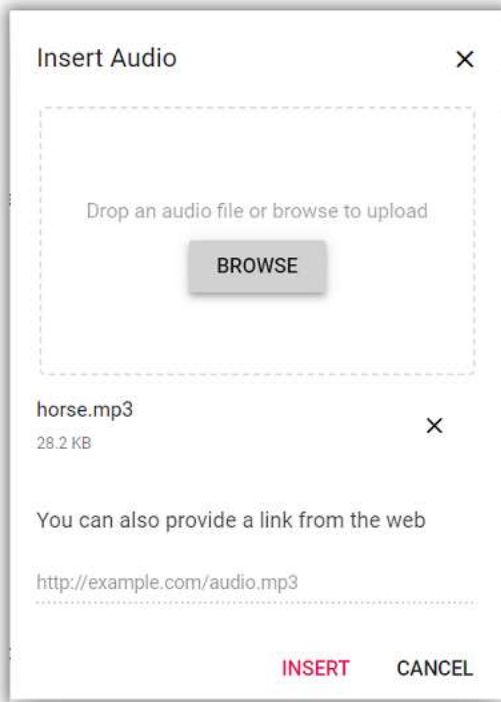
Once an audio file has been inserted, you can change it using the Rich Text Editor [quickToolbarSettings](#) `audioReplace` option. You can replace the audio file using the web URL or the browse option in the audio dialog.



Delete audio

To remove audio from the Rich Text Editor content, select the audio and click the `audioRemove` button from the quick toolbar. It will delete the audio from the Rich Text Editor content as well as from the service location if the [insertAudioSettings.removeUrl](#) is given.

Once you select the audio from the local machine, the URL for the audio will be generated. You can remove the audio from the service location by clicking the cross icon.



Display position

Sets the default display property for audio when it is inserted in the Rich Text Editor using the [insertAudioSettings.layoutOption](#) property. It has two possible options: **Inline** and **Break**. When updating the display positions, it updates the audio elements' layout position.

The default **layoutOption** property is set to **Inline**.

[Class-component]

```
`ts
import * as React from 'react';

import { HtmlEditor, Audio, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';

class App extends React.Component<{},{}> {
  private insertAudioSettings: object = {
    layoutOption: 'Inline',
  }

  public render() {
    return (
      <RichTextEditorComponent height={450} insertAudioSettings = {this.insertAudioSettings} >
        <Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]} />
      </RichTextEditorComponent>
    );
  }
}
```

```

}
}
`

```

[Functional-component]

```

`ts
import * as React from 'react';
import { HtmlEditor, Audio, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
function App() {
let insertAudioSettings: object = {
layoutOption: 'Inline',
}
return (
<RichTextEditorComponent height={450} insertAudioSettings = {insertAudioSettings} >
<Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
`

```

Rename audio before inserting

You can use the [insertAudioSettings](#) property, to specify the server handler to upload the selected audio. Then by binding the [fileUploadSuccess](#) event, you can receive the modified file name from the server and update it in the Rich Text Editor's insert audio dialog.

[Class-component]

```

`ts
import { HtmlEditor, Audio, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component<{},{}> {
private toolbarSettings: object = {
items: ['Audio']
}
private insertAudioSettings: object = {
saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/Rename",
path: "[SERVICEHOSTEDPATH]/Files/"
}
}

```

```

}
private onFileUploadSuccess (args: any) {
  alert("Get the new file name here");
  if (args.e.currentTarget.getResponseHeader('name') != null) {
    args.file.name = args.e.currentTarget.getResponseHeader('name');
    let filename: any = document.querySelectorAll(".e-file-name")[0];
    filename.innerHTML = args.file.name.replace(document.querySelectorAll(".e-file-type")[0].innerHTML,
    "");
    filename.title = args.file.name;
  }
}
public render() {
  return (
    <RichTextEditorComponent height={450} toolbarSettings={this.toolbarSettings}
    fileUploadSuccess={this.onFileUploadSuccess} insertAudioSettings={this.insertAudioSettings} >
    <Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]} />
    <p>The Rich Text Editor is WYSIWYG ("what you see is what you get") editor useful to create and edit
    content, and return the valid <a href="https://ej2.syncfusion.com/home/" target="blank">HTML
    markup</a> or <a href="https://ej2.syncfusion.com/home/" target="blank">markdown</a> of the
    content</p>
    </RichTextEditorComponent>
  );
}
export default App;

```

[Functional-component]

```

`ts
import { HtmlEditor, Audio, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App (){
  let toolbarSettings: object = {
    items: ['Audio']
  }
}

```



```

let insertAudioSettings: object = {
  saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/Rename",
  path: "[SERVICEHOSTEDPATH]/Files/"
}

function onFileUploadSuccess (args: any) {
  alert("Get the new file name here");
  if (args.e.currentTarget.getResponseHeader('name') != null) {
    args.file.name = args.e.currentTarget.getResponseHeader('name');
    let filename: any = document.querySelectorAll(".e-file-name")[0];
    filename.innerHTML = args.file.name.replace(document.querySelectorAll(".e-file-type")[0].innerHTML,
    "");
    filename.title = args.file.name;
  }
}

return (
  <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
  fileUploadSuccess={this.onFileUploadSuccess} insertAudioSettings={insertAudioSettings} >
  <Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]} />
  <p>The Rich Text Editor is WYSIWYG ("what you see is what you get") editor useful to create and edit
  content, and return the valid <a href="https://ej2.syncfusion.com/home/" target="blank">HTML
  markup</a> or <a href="https://ej2.syncfusion.com/home/" target="blank">markdown</a> of the
  content</p>
  </RichTextEditorComponent>
);
}

export default App;
`

```

To configure server-side handler, refer to the below code.

```

`csharp
int x = 0;
string file;
[AcceptVerbs("Post")]
public void Rename()
{
  try

```

```
{
var httpPostedFile = System.Web.HttpContext.Current.Request.Files["UploadFiles"];
fileName = httpPostedFile.FileName;
if (httpPostedFile != null)
{
var fileSave = System.Web.HttpContext.Current.Server.MapPath("~/Files");
if (!Directory.Exists(fileSave))
{
Directory.CreateDirectory(fileSave);
}
var fileName = Path.GetFileName(httpPostedFile.FileName);
var fileSavePath = Path.Combine(fileSave, fileName);
while (System.IO.File.Exists(fileSavePath))
{
fileName = "rteFiles" + x + "-" + fileName;
fileSavePath = Path.Combine(fileSave, fileName);
x++;
}
if (!System.IO.File.Exists(fileSavePath))
{
httpPostedFile.SaveAs(fileSavePath);
HttpResponse Response = System.Web.HttpContext.Current.Response;
Response.Clear();
Response.Headers.Add("name", fileName);
Response.ContentType = "application/json; charset=utf-8";
Response.StatusDescription = "File uploaded succesfully";
Response.End();
}
}
}
catch (Exception e)
{
HttpResponse Response = System.Web.HttpContext.Current.Response;
```

```

Response.Clear();
Response.ContentType = "application/json; charset=utf-8";
Response.StatusCode = 204;
Response.Status = "204 No Content";
Response.StatusDescription = e.Message;
Response.End();
}
}
`

```

Upload audio with authentication

You can add additional data with the audio uploaded from the Rich Text Editor on the client side, which can even be received on the server side by using the [fileUploading](#) event and its `customFormData` argument, you can pass parameters to the controller action. On the server side, you can fetch the custom headers by accessing the form collection from the current request, which retrieves the values sent using the POST method.

By default, it doesn't support the `UseDefaultCredentials` property; we need to manually append the default credentials with the upload request.

[Class-component]

```

`ts
import { HtmlEditor, Audio, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
import { UploadingEventArgs } from '@syncfusion/ej2-inputs';
class App extends React.Component<{},{}> {
  private toolbarSettings: object = {
    items: ['Audio']
  }
  private insertAudioSettings: object = {
    saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
    path: "[SERVICEHOSTEDPATH]/Files/"
  }
  private onFileUpload (args: UploadingEventArgs): void {
    var accessToken = "Authorization_token";
    // adding custom Form Data
    args.customFormData = [{ 'Authorization': accessToken }];
  }
}

```

```

}
public render() {
return (
<RichTextEditorComponent height={450} toolbarSettings={this.toolbarSettings} insertAudioSettings={
this.insertAudioSettings} fileUploading = {this.onFileUpload}>
<Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
}
,

```

[Functional-component]

```

`ts
import { HtmlEditor, Audio, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
import { UploadingEventArgs } from '@syncfusion/ej2-inputs';
function App() {
let toolbarSettings: object = {
items: ['Audio']
}
let insertAudioSettings: object = {
saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
path: "[SERVICEHOSTEDPATH]/Files/"
}
function onFileUpload (args: UploadingEventArgs): void {
var accessToken = "Authorization_token";
// adding custom Form Data
args.customFormData = [{ 'Authorization': accessToken }];
}
return (
<RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
insertAudioSettings={insertAudioSettings} fileUploading = {onFileUpload}>
<Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar]} />

```

```

</RichTextEditorComponent>
);
}
`csharp
public void SaveFiles(IList<IFormFile> UploadFiles)
{
string currentPath = Request.Form["Authorization"].ToString();
}
`

```

See Also

- [How to edit the quick toolbar settings](#)
- [How to use the link editing option in the toolbar items](#)

Video in React Rich text editor component

The Rich Text Editor allows you to insert videos from online sources and local computers and then insert them into your content. You can insert the video with the following list of options in the [insertVideoSettings](#) property.

Options	Description
----- -----	
allowedTypes	Specifies the extensions of the video types allowed to insert on bowering and passing the extensions with comma separators. For example, pass allowedTypes as <code>.mp4</code> , <code>.mov</code> , <code>.wmv</code> and <code>.avi</code> .
layoutOption	Sets the default display for a video when it is inserted into the Rich Text Editor. Possible options are: <code>Inline</code> and <code>Break</code> .
saveFormat	Sets the default save format of the video element when inserted. Possible options are: <code>Blob</code> and <code>Base64</code> .
width	Sets the default width of the video when it is inserted in the Rich Text Editor.
minWidth	Sets the minWidth of the video element when it is inserted in the Rich Text Editor.
maxWidth	Sets the maxWidth of the video element when it is inserted in the Rich Text Editor.
height	Sets the default height of the video when it is inserted in the Rich Text Editor.
minHeight	Sets the minHeight of the video element when it is inserted in the Rich Text Editor.
maxHeight	Sets the maxHeight of the video element when it is inserted in the Rich Text Editor.
saveUrl	Provides URL to map the action result method to save the video.
removeUrl	Provides URL to map the action result method to remove the video.

| path | Specifies the location to store the video. |

| resize | Sets the resizing action for the video element. |

| resizeByPercent | Sets the percentage values for the video element with the resizing action. |

Configure the video tool in the toolbar

You can add the **video** tool in the Rich Text Editor toolbar using the **toolbarSettings** [items](#) property.

Rich Text Editor features are segregated into individual feature-wise modules. To use audio, inject the **Video** module in **services**.

To configure the **Video** toolbar item, refer to the below code.

[Class-component]

APP.JSX

```
import * as React from 'react';
import { HtmlEditor, Video, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
class App extends React.Component {
  toolbarSettings = {
    items: ['Video']
  };
  insertVideoSettings = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
    removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
  };
  render() {
    return (
      <RichTextEditorComponent height={450}
        toolbarSettings={this.toolbarSettings}
        insertVideoSettings={this.insertVideoSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        <p>Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
        </ul>
      </RichTextEditorComponent>
    );
  }
}
```

```

        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]}/>
  </RichTextEditorComponent>);
}
}
export default App;

```

APP.TSX

```

import * as React from 'react';
import { HtmlEditor, Video, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
class App extends React.Component<{},{}> {
  private toolbarSettings: object = {
    items: ['Video']
  }
  private insertVideoSettings: object = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
    removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
  }
  public render() {
    return (
      <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings} insertVideoSettings =
{this.insertVideoSettings} >
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>

```

```

        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

import * as React from 'react';
import { HtmlEditor, Video, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
function App() {
  let toolbarSettings = {
    items: ['Video']
  };
  let insertVideoSettings = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
    removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
  };
  return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings} insertVideoSettings={insertVideoSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>

```



```

        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
          </li>
          <li>
            <p>Contains undo/redo manager.</p>
          </li>
          <li>
            <p>Creates bulleted and numbered lists.</p>
          </li>
        </ul>
        <Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]}/>
      </RichTextEditorComponent>);
    }
    export default App;

```

APP.TSX

```

import * as React from 'react';
import { HtmlEditor, Video, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
function App() {
  let toolbarSettings: object = {
    items: ['Video']
  }
  let insertVideoSettings: object = {
    saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
    removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
insertVideoSettings = {insertVideoSettings} >

```

```

    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
    you get") editor that provides the best user experience to create and update
    the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
        on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
        developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]} />
  </RichTextEditorComponent>
);
}
export default App;

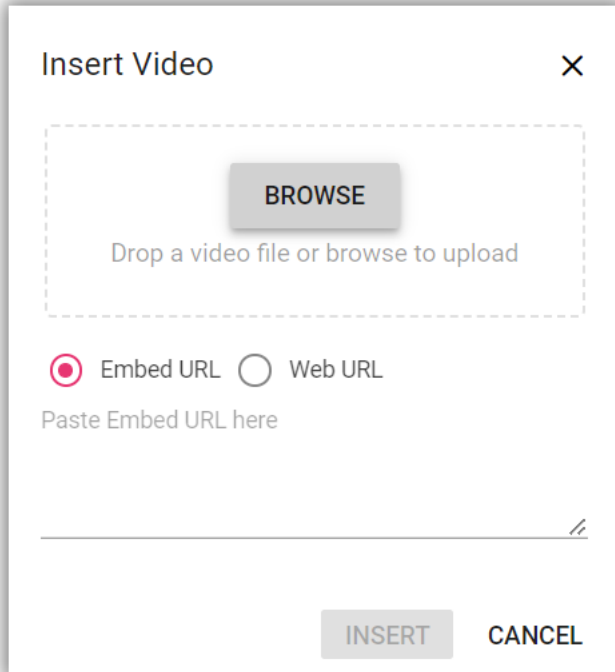
```

Insert a video from the web

You can insert a video from either the hosted link or the local machine by clicking the video button in the editor's toolbar. On Clicking the Video button, a dialog opens which allows you to insert video from the Embedded URL or web URL.

Insert from embed URL

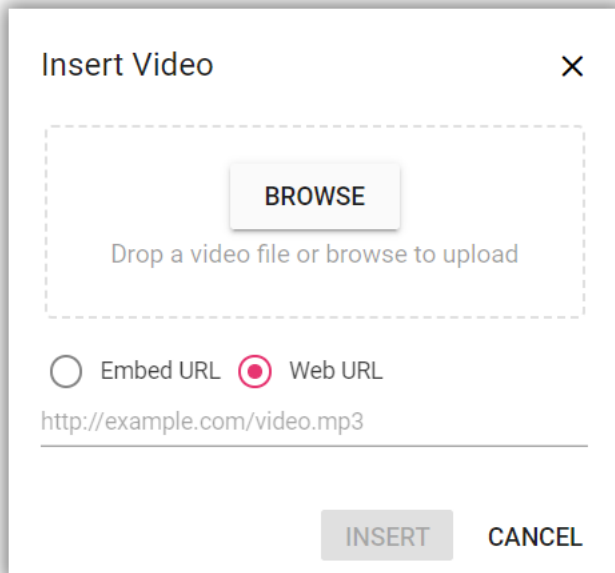
The insert video dialog opens with the **Embed URL** option as default which allows you to insert an embedded URL.



The dialog box is titled "Insert Video" with a close button (X) in the top right corner. It features a dashed border area for dropping a video file, with a "BROWSE" button and the text "Drop a video file or browse to upload". Below this, there are two radio buttons: "Embed URL" (which is selected) and "Web URL". Under the "Embed URL" option, there is a text input field with the placeholder text "Paste Embed URL here". At the bottom of the dialog, there are two buttons: "INSERT" and "CANCEL".

Insert from web URL

You can switch to **Web URL** by selecting the web URL check box. Inserting with the web URL option will add the video URL as the `src` attribute of the `<source>` tag.



The dialog box is titled "Insert Video" with a close button (X) in the top right corner. It features a dashed border area for dropping a video file, with a "BROWSE" button and the text "Drop a video file or browse to upload". Below this, there are two radio buttons: "Embed URL" and "Web URL" (which is selected). Under the "Web URL" option, there is a text input field containing the example URL "http://example.com/video.mp3". At the bottom of the dialog, there are two buttons: "INSERT" and "CANCEL".

Insert video from local machine

You can use the **browse** option on the video dialog, to select the video from the local machine and insert it into the Rich Text Editor content.

If the path field is not specified in the [insertVideoSettings](#), the video will be converted into the **Blob** URL or **Base64** and inserted inside the Rich Text Editor.

Restrict video upload based on size

You can restrict the video uploaded from the local machine when the uploaded video file size is greater than the allowed size by using the [fileUploading](#) event.

The file size in the argument will be returned in **bytes**.

In the following example, the video size has been validated before uploading and determined whether the video has been uploaded or not.

[Class-component]

```
`ts
import { HtmlEditor, Video, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
import { UploadingEventArgs } from '@syncfusion/ej2-inputs';
class App extends React.Component<{},{}> {
  private toolbarSettings: object = {
    items: ['Video']
  }
  private insertVideoSettings: object = {
    saveUrl: "https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save",
    path: "../Files/"
  }
  private onFileUpload (args: UploadingEventArgs): void {
    alert("RTE")
    let sizeInBytes: number = args.fileData.size;
    let fileSize: number = 500000;
    if (fileSize < sizeInBytes) {
      args.cancel = true;
    }
  }
  public render() {
    return (
      <RichTextEditorComponent height={450} toolbarSettings={this.toolbarSettings} insertVideoSettings={
        this.insertVideoSettings} fileUploading={this.onFileUpload}>
        <Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]} />
      </RichTextEditorComponent>
    )
  }
}
```

```

);
}
}
export default App;
`
[Functional-component]
`ts
import { HtmlEditor, Video, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
import { UploadingEventArgs } from '@syncfusion/ej2-inputs';
function App() {
  let toolbarSettings: object = {
    items: ['Video']
  }
  let insertVideoSettings: object = {
    saveUrl: "https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save",
    path: "../Files/"
  }
  function onFileUpload (args: UploadingEventArgs): void {
    alert("RTE")
    let sizeInBytes: number = args.fileData.size;
    let fileSize: number = 500000;
    if (fileSize < sizeInBytes) {
      args.cancel = true;
    }
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
    insertVideoSettings={insertVideoSettings} fileUploading={onFileUpload}>
    <Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}

```

```
export default App;
```

```
,
```

Server-side action

The selected video can be uploaded to the required destination using the controller action below. Map this method name in [insertVideoSettings.saveUrl](#) and provide the required destination path through [insertVideoSettings.path](#) properties.

If you want to insert lower-sized video files in the editor and don't want a specific physical location for saving the video, you can save the format as `Base64`.

In the following code blocks, the video module has been injected and can insert the video files saved in the specified path.

[Class-component]

```
`ts
```

```
import { HtmlEditor, Video, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
```

```
import * as React from 'react';
```

```
class App extends React.Component<{},{}> {
```

```
  private toolbarSettings: object = {
```

```
    items: ['Video']
```

```
  }
```

```
  private insertVideoSettings: object = {
```

```
    saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
```

```
    path: "[SERVICEHOSTEDPATH]/Files/"
```

```
  }
```

```
  public render() {
```

```
    return (
```

```
      <RichTextEditorComponent height={450} toolbarSettings={this.toolbarSettings}
```

```
      insertVideoSettings={this.insertVideoSettings}>
```

```
      <Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]} />
```

```
    </RichTextEditorComponent>
```

```
  );
```

```
}
```

```
}
```

```
export default App;
```

```
,
```

[Functional-component]

```

`ts
import { HtmlEditor, Video, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App(){
let toolbarSettings: object = {
items: ['Video']
}
let insertVideoSettings: object = {
saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
path: "[SERVICEHOSTEDPATH]/Files/"
}
return (
<RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
insertVideoSettings={insertVideoSettings}>
<Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
export default App;
`

`csharp
using System;
using System.IO;
using FileUpload.Models;
using System.Diagnostics;
using System.Net.Http.Headers;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Http;
using System.Collections.Generic;
using Microsoft.AspNetCore.Hosting;
namespace FileUpload.Controllers
{
public class HomeController : Controller

```

```
{
private IHostingEnvironment hostingEnv;
public HomeController(IHostingEnvironment env)
{
    hostingEnv = env;
}
public IActionResult Index()
{
    return View();
}
[AcceptVerbs("Post")]
public void SaveFiles(IList<IFormFile> UploadFiles)
{
    try
    {
        foreach (IFormFile file in UploadFiles)
        {
            if (UploadFiles != null)
            {
                string filename = ContentDispositionHeaderValue.Parse(file.ContentDisposition).FileName.Trim("");
                filename = hostingEnv.WebRootPath + "\\Files" + $"@\"{filename}\"";
                // Create a new directory, if it does not exists
                if (!Directory.Exists(hostingEnv.WebRootPath + "\\Files"))
                {
                    Directory.CreateDirectory(hostingEnv.WebRootPath + "\\Files");
                }
                if (!System.IO.File.Exists(filename))
                {
                    using (FileStream fs = System.IO.File.Create(filename))
                    {
                        file.CopyTo(fs);
                        fs.Flush();
                    }
                }
            }
        }
    }
}
```



```

Response.StatusCode = 200;
}
}
}
}
catch (Exception)
{
Response.StatusCode = 204;
}
}
[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error()
{
return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
}
}
}
}
`

```

Video save format

The video files can be saved as **Blob** or **Base64** URL by using the [insertVideoSettings.saveFormat](#) property, which is of enum type, and the generated URL will be set to the **src** attribute of the **<source>** tag.

The default **saveFormat** property is set to **Blob** format.

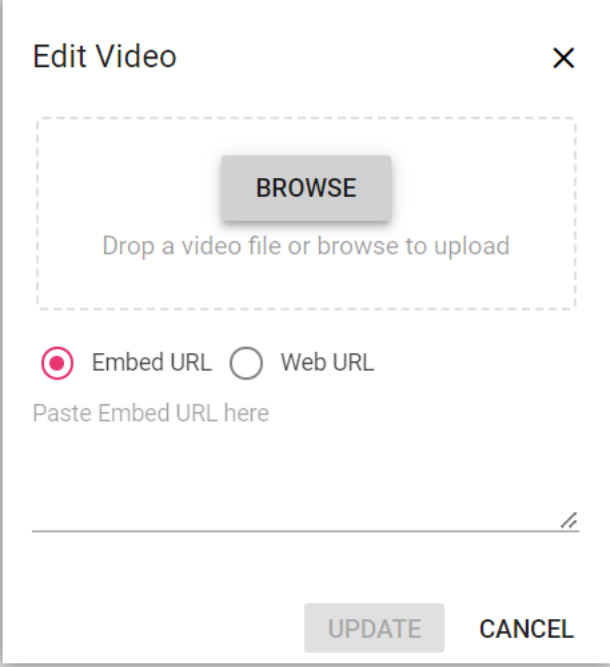
```

`html
<video>
<source src="blob:http://ej2.syncfusion.com/3ab56a6e-ec0d-490f-85a5-f0aeb0ad8879"
type="video/mp4" >
</video>
<video>
<source src="data:video/mp4;base64,iVBORw0KGgoAAAANSUhEUgAAADAAAAAwCAYAAABXAvmHA"
type="video/mp4" >
</video>
`

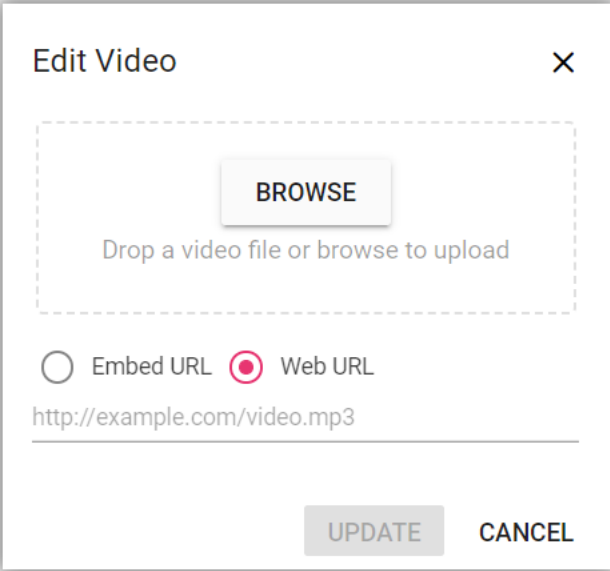
```

Replacing video

Once a video file has been inserted, you can replace it using the Rich Text Editor [quickToolbarSettings](#) `videoReplace` option. You can replace the video file either by using the embedded URL or the web URL and the browse option in the video dialog.



The 'Edit Video' dialog box features a close button (X) in the top right corner. It contains a dashed rectangular area with a 'BROWSE' button and the text 'Drop a video file or browse to upload'. Below this, there are two radio buttons: 'Embed URL' (which is selected) and 'Web URL'. A text input field is positioned below the radio buttons with the placeholder text 'Paste Embed URL here'. At the bottom of the dialog are 'UPDATE' and 'CANCEL' buttons.

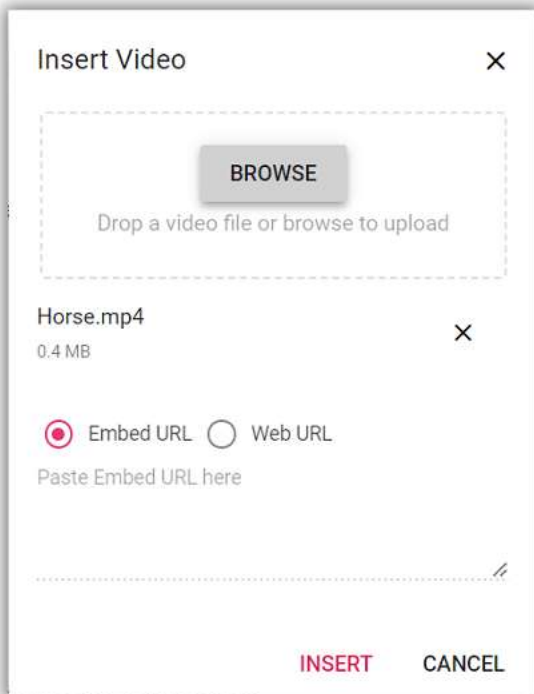


The 'Edit Video' dialog box is identical to the one above, but with the 'Web URL' radio button selected. The text input field below the radio buttons contains the example URL 'http://example.com/video.mp3'. The 'UPDATE' and 'CANCEL' buttons are at the bottom.

Delete video

To remove a video from the Rich Text Editor content, select the video and click the `videoRemove` button from the quick toolbar. It will delete the video from the Rich Text Editor content as well as from the service location if the [insertVideoSettings.removeUrl](#) is given.

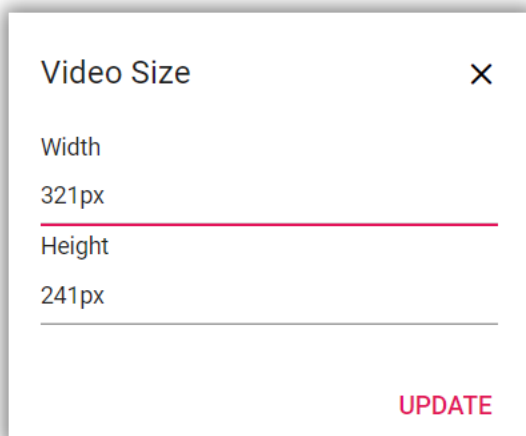
Once you select the video from the local machine, the URL for the video will be generated. You can remove the video from the service location by clicking the cross icon.



Dimension

Set the default width, minWidth, height, and minHeight of the video element, when it is inserted in the Rich Text Editor using the [width](#), [minWidth](#), [height](#), [minHeight](#) properties.

Through the [quickToolbarSettings](#), also you can change the width and height using the **Change Size** button. Once you click on the button, the video size dialog will open as below. In that, specify the width and height of the video in pixels.



Display position

Sets the default display property for the video when it is inserted in the Rich Text Editor using the [insertVideoSettings.layoutOption](#) property. It has two possible options: **Inline** and **Break**. When updating the display positions, it updates the video elements' layout position.

The default **layoutOption** property is set to **Inline**.

[Class-component]

```
`ts
import * as React from 'react';

import { HtmlEditor, Video, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';

class App extends React.Component<{},{}> {
  private insertVideoSettings: object = {
    layoutOption: 'Inline',
  }

  public render() {
    return (
      <RichTextEditorComponent height={450} insertVideoSettings = {this.insertVideoSettings} >
        <Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]} />
      </RichTextEditorComponent>
    );
  }
}
```

[Functional-component]

```
`ts
import * as React from 'react';

import { HtmlEditor, Video, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';

function App(){
  let insertVideoSettings: object = {
    layoutOption: 'Inline',
  }

  return (
    <RichTextEditorComponent height={450} insertVideoSettings = {insertVideoSettings} >
```

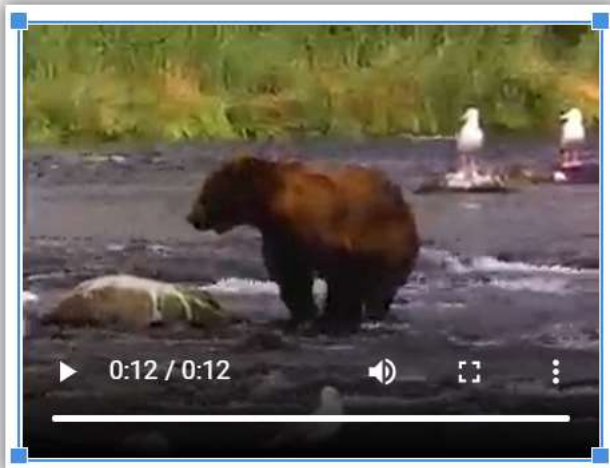
```
<Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
`
```

Resize video

The Rich Text Editor has built-in video resizing support, which is enabled for the video elements added. The resize points will appear on each corner of the video when focusing so users can easily resize the video using mouse points or thumb through the resize points. Also, the resize calculation will be done based on the aspect ratio.

You can disable the resize action by configuring `false` for the [insertVideoSettings.resize](#) property.

If the [minWidth](#) and [minHeight](#) properties are configured, the video resizing does not shrink below the specified values.



Rename video before inserting

You can use the [insertVideoSettings](#) property, to specify the server handler to upload the selected video. Then by binding the [fileUploadSuccess](#) event, you can receive the modified file name from the server and update it in the Rich Text Editor's insert video dialog.

[Class-component]

```
`ts
```

```
import { HtmlEditor, Video, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
```

```
import * as React from 'react';
```

```
class App extends React.Component<{},{}> {
```

```
  private toolbarSettings: object = {
```

```
    items: ['Video']
```

```

}
private insertVideoSettings: object = {
  saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/Rename",
  path: "[SERVICEHOSTEDPATH]/Files/"
}
private onFileUploadSuccess (args: any) {
  alert("Get the new file name here");
  if (args.e.currentTarget.getResponseHeader('name') != null) {
    args.file.name = args.e.currentTarget.getResponseHeader('name');
    let filename: any = document.querySelectorAll(".e-file-name")[0];
    filename.innerHTML = args.file.name.replace(document.querySelectorAll(".e-file-type")[0].innerHTML,
    "");
    filename.title = args.file.name;
  }
}
public render() {
  return (
    <RichTextEditorComponent height={450} toolbarSettings={this.toolbarSettings}
    fileUploadSuccess={this.onFileUploadSuccess} insertVideoSettings={this.insertVideoSettings} >
    <Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]} />
    <p>The Rich Text Editor is WYSIWYG ("what you see is what you get") editor useful to create and edit
    content, and return the valid <a href="https://ej2.syncfusion.com/home/" target="blank">HTML
    markup</a> or <a href="https://ej2.syncfusion.com/home/" target="blank">markdown</a> of the
    content</p>
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

```

`ts
import { HtmlEditor, Video, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
'@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';

```

```

function App(){
let toolbarSettings: object = {
items: ['Video']
}
let insertVideoSettings: object = {
saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/Rename",
path: "[SERVICEHOSTEDPATH]/Files/"
}
function onFileUploadSuccess (args: any) {
alert("Get the new file name here");
if (args.e.currentTarget.getResponseHeader('name') != null) {
args.file.name = args.e.currentTarget.getResponseHeader('name');
let filename: any = document.querySelectorAll(".e-file-name")[0];
filename.innerHTML = args.file.name.replace(document.querySelectorAll(".e-file-type")[0].innerHTML,
"");
filename.title = args.file.name;
}
}
return (
<RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
fileUploadSuccess={onFileUploadSuccess} insertVideoSettings={insertVideoSettings} >
<Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]} />
<p>The Rich Text Editor is WYSIWYG ("what you see is what you get") editor useful to create and edit
content, and return the valid <a href="https://ej2.syncfusion.com/home/" target="blank">HTML
markup</a> or <a href="https://ej2.syncfusion.com/home/" target="blank">markdown</a> of the
content</p>
</RichTextEditorComponent>
);
}
export default App;

```

To configure server-side handler, refer to the below code.

```

`csharp
int x = 0;
string file;

```

```
[AcceptVerbs("Post")]
public void Rename()
{
    try
    {
        var httpPostedFile = System.Web.HttpContext.Current.Request.Files["UploadFiles"];
        fileName = httpPostedFile.FileName;
        if (httpPostedFile != null)
        {
            var fileSave = System.Web.HttpContext.Current.Server.MapPath("~/Files");
            if (!Directory.Exists(fileSave))
            {
                Directory.CreateDirectory(fileSave);
            }
            var fileName = Path.GetFileName(httpPostedFile.FileName);
            var fileSavePath = Path.Combine(fileSave, fileName);
            while (System.IO.File.Exists(fileSavePath))
            {
                fileName = "rteFiles" + x + "-" + fileName;
                fileSavePath = Path.Combine(fileSave, fileName);
                x++;
            }
            if (!System.IO.File.Exists(fileSavePath))
            {
                httpPostedFile.SaveAs(fileSavePath);
                HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
                Response.Clear();
                Response.Headers.Add("name", fileName);
                Response.ContentType = "application/json; charset=utf-8";
                Response.StatusDescription = "File uploaded succesfully";
                Response.End();
            }
        }
    }
}
```



```

}
catch (Exception e)
{
    HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
    Response.Clear();
    Response.ContentType = "application/json; charset=utf-8";
    Response.StatusCode = 204;
    Response.Status = "204 No Content";
    Response.StatusDescription = e.Message;
    Response.End();
}
}
,

```

Upload video with authentication

You can add additional data with the video uploaded from the Rich Text Editor on the client side, which can even be received on the server side by using the [fileUploading](#) event and its [customFormData](#) argument, you can pass parameters to the controller action. On the server side, you can fetch the custom headers by accessing the form collection from the current request, which retrieves the values sent using the POST method.

By default, it doesn't support the [UseDefaultCredentials](#) property, you can manually append the default credentials with the upload request.

[Class-component]

```

`ts
import { HtmlEditor, Video, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
import { UploadingEventArgs } from '@syncfusion/ej2-inputs';
class App extends React.Component<{},{}> {
    private toolbarSettings: object = {
        items: ['Video']
    }
    private insertVideoSettings: object = {
        saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
        path: "[SERVICEHOSTEDPATH]/Files/"
    }
}

```

```

private onFileUpload (args: UploadingEventArgs): void {
  var accessToken = "Authorization_token";
  // adding custom Form Data
  args.customFormData = [{ 'Authorization': accessToken }];
}

public render() {
  return (
    <RichTextEditorComponent height={450} toolbarSettings={this.toolbarSettings} insertVideoSettings={
      this.insertVideoSettings} fileUploading = {this.onFileUpload}>
      <Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}

```

[Functional-component]

```

`ts
import { HtmlEditor, Video, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
import { UploadingEventArgs } from '@syncfusion/ej2-inputs';
function App(){
  let toolbarSettings: object = {
    items: ['Video']
  }
  let insertVideoSettings: object = {
    saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
    path: "[SERVICEHOSTEDPATH]/Files/"
  }
  let onFileUpload (args: UploadingEventArgs): void {
    var accessToken = "Authorization_token";
    // adding custom Form Data
    args.customFormData = [{ 'Authorization': accessToken }];

```

```

}
return (
  <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
  insertVideoSettings={insertVideoSettings} fileUploading = {onFileUpload}>
    <Inject services={[Toolbar, Video, Link, HtmlEditor, QuickToolbar]} />
  </RichTextEditorComponent>
);
}
`csharp
public void SaveFiles(IList<IFormFile> UploadFiles)
{
  string currentPath = Request.Form["Authorization"].ToString();
}

```

See Also

- [How to edit the quick toolbar settings](#)
- [How to use the link editing option in the toolbar items](#)

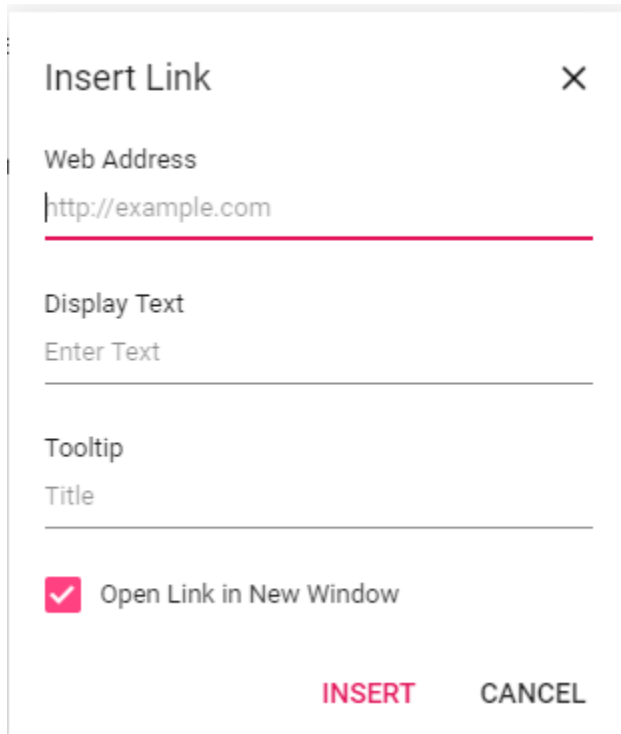
Link in React Rich text editor component

A hyperlink can be insert into the editor for quick access to the related information. The hyperlink itself can be a text or an image.

Insert link

Point the cursor anywhere within the editor where you would like to insert the link. It is also possible to select a text or an image within the editor and can be converted to the hyperlink. Click the Insert HyperLink tool on the toolbar. The Insert Link Dialog will be open. The dialog has the following options.

Rich Text Editor features are segregated into individual feature-wise modules. To use image and link tool, inject link module using the `RichTextEditor.Inject(link)`.



The image shows a modal dialog box titled "Insert Link" with a close button (X) in the top right corner. It contains three input fields: "Web Address" with the text "http://example.com", "Display Text" with the placeholder "Enter Text", and "Tooltip" with the placeholder "Title". Below these fields is a checkbox labeled "Open Link in New Window" which is checked. At the bottom right are two buttons: "INSERT" in red and "CANCEL" in gray.

| Options | Description |

| --- | --- |

| Web Address | Types or paste the destination for the link you are creating. |

| Display Text | Types or edit the required text that you want to display text for the link. |

| Tooltip | Displays additional helpful information when you place the pointer on the hyperlink, type the required text in the "Tooltip" field. |

| Open Link in New Window | Specifies whether the given link will be open in new window or not. |

The Rich Text Editor link tool validates the URLs, as you type in the Web Address. URLs considered invalid will be highlighted with red color by clicking the insert button in the **Insert Link** dialog.

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - Insert Link Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['CreateLink']
  };
  render() {
    return (<RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}>
```

```

    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
    you get") editor that provides the best user experience to create and update
    the content.
      Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
  </RichTextEditorComponent>;
}
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Insert Link Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {

```

```

private toolbarSettings: object = {
  items: ['CreateLink']
}
public render() {
  return (
    <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Insert Link Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    let toolbarSettings = {
        items: ['CreateLink']
    };
    return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
        <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
            <p>Capable of handling markdown editing.</p>
        </li>
        <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
            <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
            <p>Supports third-party library integration.</p>
        </li>
        <li>
            <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
</RichTextEditorComponent>);
}

```

```
export default App;
```

APP.TSX

```
/**
 * Rich Text Editor - Insert Link Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings: object = {
    items: ['CreateLink']
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
    </RichTextEditorComponent>
  );
}
```



```

    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
  </RichTextEditorComponent>
);
}
export default App;

```

Remove link

If you want to remove a hyperlink from a text or image, select the text or image with the hyperlink and click Remove Hyperlink tool from the toolbar. It will keep the text or image.

Auto-link

When you type URL, and Enter key to the Rich Text Editor, the typed URL will be automatically changed into the hyperlink.

Manipulation

Add the custom tools on the selected link inside the Rich Text Editor through the quick toolbar.



The quick toolbar for the link has the following options.

Name	Description
Open	The given link page, will be open in new window.
Edit Link	Used to edit the link in the Rich Text Editor content.
Remove Link	Used to remove link from the content of Rich Text Editor.
Custom Tool	Used to add the custom options in the quick toolbar.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - Link Manipulation Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['CreateLink']
  };
  render() {
    return (
      <RichTextEditorComponent height={450}
        toolbarSettings={this.toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides the best user experience to create and update the content.
          Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
      </RichTextEditorComponent>
    );
  }
}

```

```

        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
          </li>
          <li>
            <p>Contains undo/redo manager.</p>
          </li>
          <li>
            <p>Creates bulleted and numbered lists.</p>
          </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
      </RichTextEditorComponent>;
    }
  }
  export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Link Manipulation Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private toolbarSettings: object = {
    items: ['CreateLink']
  }
  public render() {
    return (

```

```

    <RichTextEditorComponent height={450}
    toolbarSettings={this.toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Link Manipulation Sample

```

```

*/
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['CreateLink']
  };
  return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings}>
  <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
  Users can format their content using standard toolbar commands.</p>
  <p><b>Key features:</b></p>
  <ul>
    <li>
      <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
    </li>
    <li>
      <p>Capable of handling markdown editing.</p>
    </li>
    <li>
      <p>Contains a modular library to load the necessary functionality
on demand.</p>
    </li>
    <li>
      <p>Provides a fully customizable toolbar.</p>
    </li>
    <li>
      <p>Provides HTML view to edit the source directly for
developers.</p>
    </li>
    <li>
      <p>Supports third-party library integration.</p>
    </li>
    <li>
      <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
      <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
    </li>
    <li>
      <p>Contains undo/redo manager.</p>
    </li>
    <li>
      <p>Creates bulleted and numbered lists.</p>
    </li>
  </ul>
  <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>
  </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Link Manipulation Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings: object = {
    items: ['CreateLink']
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}
export default App;

```

Table in React Rich text editor component

Rich Text Editor allows to insert table of content in edit panel and provide options to add, edit, and remove the table as well as perform other table related action. For inserting the table to the Rich Text Editor, the following list of options have been provided in the [tableSettings](#)

Options	Description	Default Value
minWidth	Sets the default minWidth of the table.	0
maxWidth	Sets the default maxWidth of the table.	null
resize	Enable resize feature in table.	true
styles	This is an array of key value pair, on each pair, key should be name of styling and value is class name. this list will be shown on quick toolbar options to change the styles of table on designing like dashed, double bordered.	TableStyleItems
width	Sets the default width of the table.	100%

Rich Text Editor features are segregated into individual feature-wise modules. To use table tool, inject table module using the `<Inject services=[[Table]] />`.

Insert table

Using the **table** toolbar option, select a number of rows and columns to be inserted over the table grid and insert table into Rich Text Editor content using the mouse.

Tables can also be inserted through the **Insert Table** option in the pop-up where the number of rows and columns can be provided manually, and this is the default way in devices.

In the following sample, the table has been injected from table module.

[Class-component]

APP.JSX

```
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Table, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['CreateTable']
  };
  render() {
    return (<RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>

```

```

        </li>
        <li>
            <p>Capable of handling markdown editing.</p>
        </li>
        <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
            <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
            <p>Supports third-party library integration.</p>
        </li>
        <li>
            <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Table]}/>
    </RichTextEditorComponent>;
    }
}
export default App;

```

APP.TSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Table, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    private toolbarSettings: object = {
        items: ['CreateTable']
    }
    public render() {
        return (
            <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}>
                <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
            </p>
            </RichTextEditorComponent>
        );
    }
}

```

```

    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Table]} />
  </RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Table, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['CreateTable']
  }

```



```

    };
    return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
    <li>
    <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
    </li>
    <li>
    <p>Capable of handling markdown editing.</p>
    </li>
    <li>
    <p>Contains a modular library to load the necessary functionality
on demand.</p>
    </li>
    <li>
    <p>Provides a fully customizable toolbar.</p>
    </li>
    <li>
    <p>Provides HTML view to edit the source directly for
developers.</p>
    </li>
    <li>
    <p>Supports third-party library integration.</p>
    </li>
    <li>
    <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
    <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
    </li>
    <li>
    <p>Contains undo/redo manager.</p>
    </li>
    <li>
    <p>Creates bulleted and numbered lists.</p>
    </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Table]}/>
    </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Table, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    let toolbarSettings: object = {

```

```

    items: ['CreateTable']
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides the best user experience to create and update the content.
        Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar, Table]} />
    </RichTextEditorComponent>
  );
}
export default App;

```

Quick Toolbar

Quick toolbar is opened by clicking the table. It has different sets of commands to be performed on the table which increases the feasibility to edit the table easily.

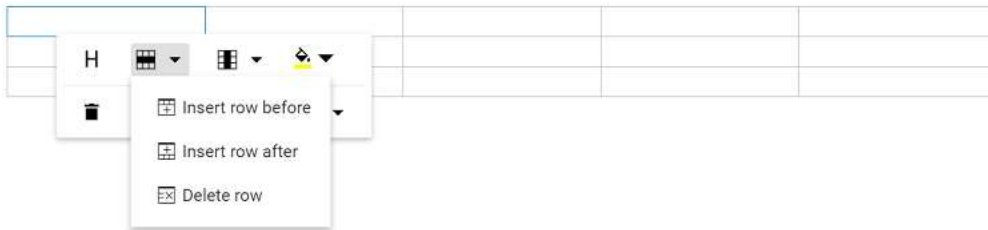
To use quick toolbar, inject the quick toolbar module using the `<Inject services={[QuickToolbar]} />`.

Table Header

Table Header command is available with quick toolbar option through which the header row can be added or removed from the inserted table. The following image illustrates the table header.

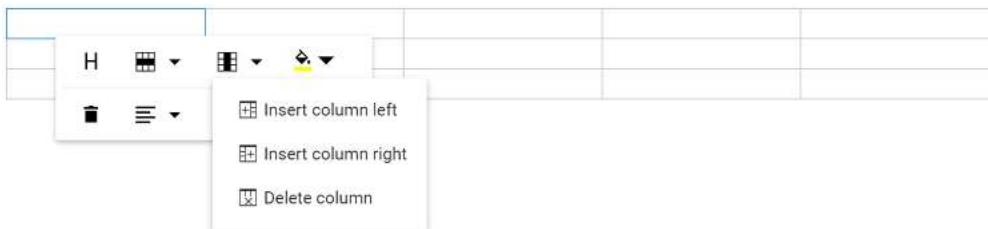
Insert Rows

Rows can be inserted above or below the required table cell through the quick toolbar. Also, focused row can be deleted. The following screenshot shows the available options of the row item.



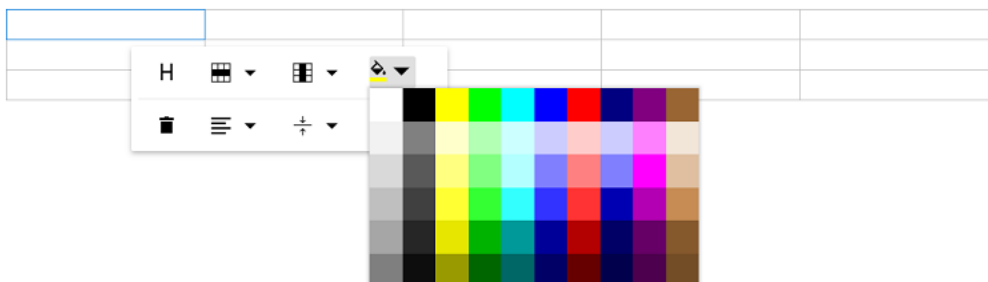
Insert Columns

Columns can be inserted to the left or right side of the required table cell through the quick toolbar. Also, the focused column can be deleted. The following screenshot shows the available options of the column item.



Set Color

The background color can be set for each table cell through the **background color** command available with quick toolbar.

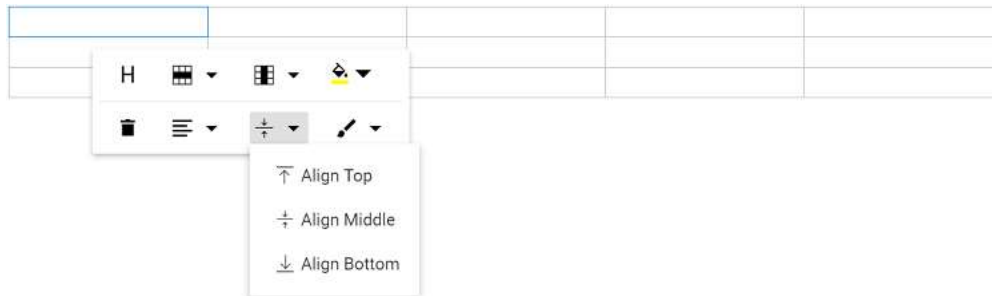


Delete Table

Using the delete item in the quick toolbar, users can delete the entire table.

Vertical Align

Text inside the table can be aligned to top, middle, or bottom using the **tableCellVerticalAlign** tool of the quick toolbar.



Horizontal Align

Text inside the table can be aligned left, right, or center using the `tableCellHorizontalAlign` tool of the quick toolbar.

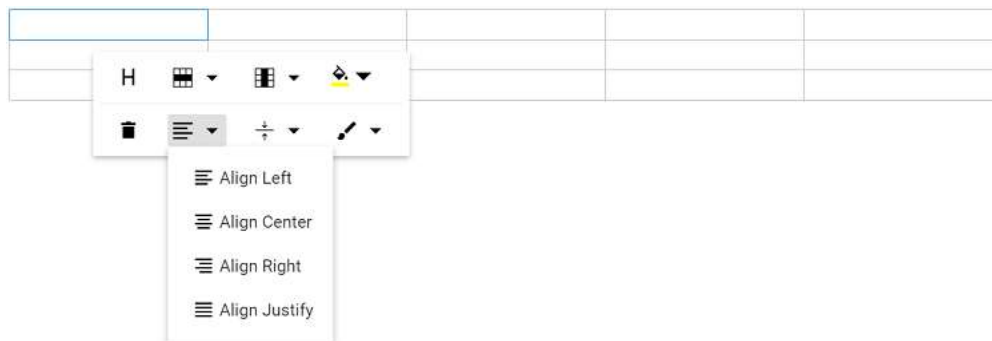


Table Styles

Table styles provided for class name should be appended to a table element. It helps to design the table in specific CSS styles when inserting in the editor.

By Default, provides Dashed border and Alternate rows.

Dashed border: Applies the dashed border to the table.

Alternate border: Applies the alternative background to the table.

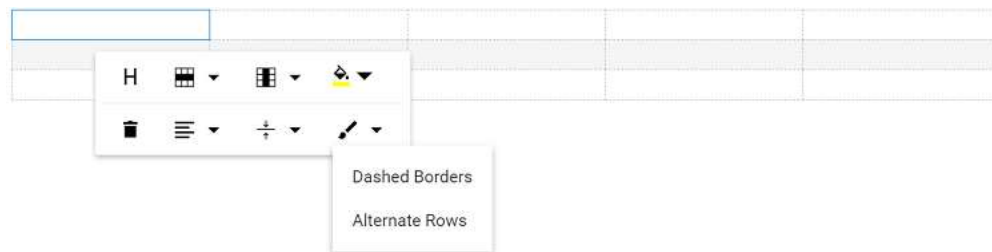


Table Properties

Sets the default width of the table when it is inserted in the Rich Text Editor using the width of `tableSettings`.

Using the quick toolbar, users can change the width, cell padding, and cell spacing in the selected table using the properties option.

Edit Table

×

Width

1,189

▼ ▲

Cell Padding

0

▼ ▲

Cell Spacing

0

▼ ▲

UPDATE

CANCEL

Table cell merge and split

The Rich Text Editor allows users to change the appearance of the tables by splitting or merging the table cells.

TableCell item should be configured in the Table [quickToolbarSettings](#) Property to show the merge/split icons while selecting the table cells

Table cell merge

The table cell merge feature allows you to merge two or more row and column cells into a single cell with its contents.

The following image explains the table merge action.

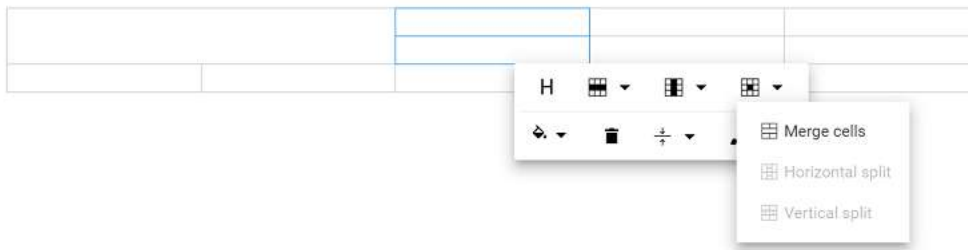


Table cell split

The table cell split feature allows you to a selected cell can be split both horizontally and vertically.

The following image explains the table split action.



[Class-component]**APP.JSX**

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Table, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['CreateTable']
  };
  quickToolbarSettings = {
    table: ['TableHeader', 'TableRows', 'TableColumns', 'TableCell', '-',
'BackgroundColor', 'TableRemove', 'TableCellVerticalAlign', 'Styles']
  };
  render() {
    return (<RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}
quickToolbarSettings={this.quickToolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
          </li>
          <li>
            <p>Contains undo/redo manager.</p>
          </li>
          <li>

```

```

        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Table]}/>
  </RichTextEditorComponent>);
}
}
export default App;

```

APP.TSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Table, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private toolbarSettings: object = {
    items: ['CreateTable']
  }
  private quickToolbarSettings: object = {
    table: ['TableHeader', 'TableRows', 'TableColumns', 'TableCell', '-',
'BackgroundColor', 'TableRemove', 'TableCellVerticalAlign', 'Styles']
  }
  public render() {
    return (
      <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}
quickToolbarSettings={this.quickToolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>

```

```

        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Table]} />
  </RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Table, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['CreateTable']
  };
  let quickToolbarSettings = {
    table: ['TableHeader', 'TableRows', 'TableColumns', 'TableCell', '-',
'BackgroundColor', 'TableRemove', 'TableCellVerticalAlign', 'Styles']
  };
  return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings}
quickToolbarSettings={quickToolbarSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
    </ul>
  </RichTextEditorComponent>
);
}
export default App;

```



```

        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Table]}/>
    </RichTextEditorComponent>);
  }
  export default App;

```

APP.TSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Table, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings: object = {
    items: ['CreateTable']
  }
  let quickToolbarSettings: object = {
    table: ['TableHeader', 'TableRows', 'TableColumns', 'TableCell', '-',
'BackgroundColor', 'TableRemove', 'TableCellVerticalAlign', 'Styles']
  }
  return (
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
quickToolbarSettings={quickToolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>

```

```

        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Table]} />
  </RichTextEditorComponent>
);
}
export default App;

```

Emoji Picker in React RichTextEditor Component

An emoji picker is a tool that allows users to add emojis or emoticons to their text easily. Typically, it is a small window or panel that displays a variety of emojis arranged in different categories, such as smileys, animals, food, and so on. Users can select the desired emoji by clicking on it or by typing its name in a search bar.

Enabling the toolbar option and custom emojis.

Add the `EmojiPicker` tool to the toolbar of the `RichTextEditor` by utilizing the `toolbarSettings` [items](#) property.

By default, a predefined set of emojis is configured. However, you can customize these icons according to your needs. To achieve this, utilize the [emojiPickerSettings](#) property.

```
`ts
```

```
import { RichTextEditorComponent, Inject, Toolbar, HtmlEditor, Image, QuickToolbar, Link, EmojiPicker }
from '@syncfusion/ej2-react-richtexteditor';
```

```
import * as React from 'react';
```

```
class App extends React.Component<{},{}> {
  private toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', '|', 'Formats', 'Alignments', 'OrderedList',
    'UnorderedList', '|', 'CreateLink', 'Image', '|', 'SourceCode', 'EmojiPicker', '|', 'Undo', 'Redo'
  ]
}

  private emojiPickerSettings: object = {
    iconsSet: [{name: 'Smilies & People', code: '1F600', iconCss: 'e-emoji',
    icons: [{ code: '1F600', desc: 'Grinning face' },
    { code: '1F603', desc: 'Grinning face with big eyes' },
    { code: '1F604', desc: 'Grinning face with smiling eyes' },
    { code: '1F606', desc: 'Grinning squinting face' },
    { code: '1F605', desc: 'Grinning face with sweat' },
    { code: '1F602', desc: 'Face with tears of joy' },
    { code: '1F923', desc: 'Rolling on the floor laughing' },
    { code: '1F60A', desc: 'Smiling face with smiling eyes' } ]
  }, {
    name: 'Animals & Nature', code: '1F435', iconCss: 'e-animals',
    icons: [{ code: '1F436', desc: 'Dog face' },
    { code: '1F431', desc: 'Cat face' },
    { code: '1F42D', desc: 'Mouse face' },
    { code: '1F439', desc: 'Hamster face' },
    { code: '1F430', desc: 'Rabbit face' },
    { code: '1F98A', desc: 'Fox face' } ]
  }, {
    name: 'Food & Drink', code: '1F347', iconCss: 'e-food-and-drinks',
    icons: [{ code: '1F34E', desc: 'Red apple' },
    { code: '1F34C', desc: 'Banana' },
    { code: '1F347', desc: 'Grapes' },
    { code: '1F353', desc: 'Strawberry' },
    { code: '1F35E', desc: 'Bread' },
    { code: '1F950', desc: 'Croissant' },
    { code: '1F955', desc: 'Carrot' },
```

```
{ code: '1F354', desc: 'Hamburger' }}
}, {
name: 'Activities', code: '1F383', iconCss: 'e-activities',
icons: [{ code: '26BD', desc: 'Soccer ball' },
{ code: '1F3C0', desc: 'Basketball' },
{ code: '1F3C8', desc: 'American football' },
{ code: '26BE', desc: 'Baseball' },
{ code: '1F3BE', desc: 'Tennis' },
{ code: '1F3D0', desc: 'Volleyball' },
{ code: '1F3C9', desc: 'Rugby football' }]
}, {
name: 'Travel & Places', code: '1F30D', iconCss: 'e-travel-and-places',
icons: [{ code: '2708', desc: 'Airplane' },
{ code: '1F697', desc: 'Automobile' },
{ code: '1F695', desc: 'Taxi' },
{ code: '1F6B2', desc: 'Bicycle' },
{ code: '1F68C', desc: 'Bus' }]
}, {
name: 'Objects', code: '1F507', iconCss: 'e-objects', icons: [{ code: '1F4A1', desc: 'Light bulb' },
{ code: '1F526', desc: 'Flashlight' },
{ code: '1F4BB', desc: 'Laptop computer' },
{ code: '1F5A5', desc: 'Desktop computer' },
{ code: '1F5A8', desc: 'Printer' },
{ code: '1F4F7', desc: 'Camera' },
{ code: '1F4F8', desc: 'Camera with flash' },
{ code: '1F4FD', desc: 'Film projector' }]
}, {
name: 'Symbols', code: '1F3E7', iconCss: 'e-symbols', icons: [{ code: '274C', desc: 'Cross mark' },
{ code: '2714', desc: 'Check mark' },
{ code: '26A0', desc: 'Warning sign' },
{ code: '1F6AB', desc: 'Prohibited' },
{ code: '2139', desc: 'Information' },
{ code: '267B', desc: 'Recycling symbol' },
```

```

{ code: '1F6AD', desc: 'No smoking' }}
}}
}

public render() {
return (
<RichTextEditorComponent height={450} toolbarSettings={this.toolbarSettings} emojiPickerSettings={
this.emojiPickerSettings}>
<Inject services={[Toolbar, Audio, Link, HtmlEditor, QuickToolbar, EmojiPicker]} />
</RichTextEditorComponent>
);
}
}

export default App;
`

```

Additionally, you have the option to customize the icons of toolbar items using the [iconCSS](#) and [code](#) properties. The `iconCSS` property allows you to define a custom CSS class for the toolbar item icon, while the `code` property enables you to specify the Unicode character code for the icon.

When both `iconCSS` and `code` properties are provided, the `iconCSS` property takes precedence in determining the appearance of the toolbar item icon.

Additionally, you have the option to enhance the user experience by implementing a filtering feature for efficiently managing a large dataset of emojis. By setting the [showSearchBox](#) property to true (which is the default value), users will be able to utilize a search box to filter the displayed emojis according to their preferences.

The following code example shows how to add the emoji picker tool in the RichTextEditor.

APP.JSX

```

import * as React from 'react';
import { RichTextEditorComponent, Inject, Toolbar, HtmlEditor, Image,
QuickToolbar, Link, EmojiPicker } from '@syncfusion/ej2-react-
richtexteditor';
class App extends React.Component {
  toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', '|', 'Formats', 'Alignments',
'OrderedList',
'UnorderedList', '|', 'CreateLink', 'Image', '|', 'SourceCode',
'EmojiPicker', '|', 'Undo', 'Redo']
  };
  render() {
    return (
      <RichTextEditorComponent height={450}
      toolbarSettings={this.toolbarSettings}>
        <p>An emoji picker in a Rich Text Editor is a tool that allows
users to easily add emojis or emoticons to their text.</p>

```

```

        <p>Typically, it is a small window or panel that displays a variety
of emojis, arranged in different categories, such as smileys, animals, food,
and so on. Users can select the desired emoji by clicking on it or by typing
its name in a search bar.</p>
        <Inject services={[Toolbar, HtmlEditor, Image, QuickToolbar, Link,
EmojiPicker]} />
    </RichTextEditorComponent>);
    }
}
export default App;

```

APP.TSX

```

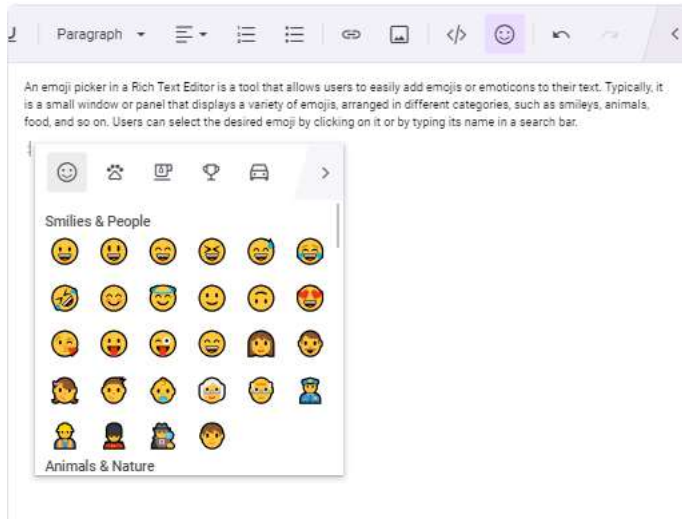
import * as React from 'react';
import { RichTextEditorComponent, Inject, Toolbar, HtmlEditor, Image,
QuickToolbar, Link, EmojiPicker } from '@syncfusion/ej2-react-
richtexteditor';
class App extends React.Component<{}, {}> {
    private toolbarSettings: object = {
        items: ['Bold', 'Italic', 'Underline', '|', 'Formats', 'Alignments',
'OrderedList',
'UnorderedList', '|', 'CreateLink', 'Image', '|', 'SourceCode',
'EmojiPicker', '|', 'Undo', 'Redo']
    }
    public render() {
        return (
            <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings} >
                <p>An emoji picker in a Rich Text Editor is a tool that allows users
to easily add emojis or emoticons to their text.</p>
                <p>Typically, it is a small window or panel that displays a variety
of emojis, arranged in different categories, such as smileys, animals, food,
and so on. Users can select the desired emoji by clicking on it or by typing
its name in a search bar.</p>
                <Inject services={[Toolbar, HtmlEditor, Image, QuickToolbar, Link,
EmojiPicker]} />
            </RichTextEditorComponent>
        );
    }
}
export default App;

```

Rich Text Editor features are segregated into individual feature-wise modules. To use emojis, inject the `EmojiPicker` module in `services`.

Using the shortcut key to open the emoji picker

Quickly access the emoji picker by pressing the colon (:) key while typing a word prefix in an editor, allowing instant emoji selection and display. Moreover, continue typing in the editor after the colon (:) to filter and refine your search for the desired emojis.



Navigating and selecting emojis using the keyboard

The emoji picker popup offers keyboard navigation options, allowing you to move the emoji focus from one emoji to another. The following keys are used for navigation:

Arrow keys: Use the arrow keys (up, down, left, right) to move the emoji focus in the corresponding direction.

Enter: Press Enter key to select the currently focused emoji.

Escape: Press Escape to close the emoji picker popup without selecting an emoji.

Markdown in React Rich text editor component

When you format the word in Markdown format, you should add Markdown syntax to the word to indicate the words and phrases that looks different from each other.

Rich Text Editor supports markdown editing when the `editorMode` set as **markdown** and using both *keyboard interaction* and *toolbar action*, you can apply the formatting to text.

To create Rich Text Editor with Markdown editing feature, inject the MarkdownEditor module to the RTE using the `RichTextEditor.Inject(MarkdownEditor)` method.

Supported Commands

The React Markdown editor supports the following commands to format the markdown content:

Commands	Syntax	Description
----------	--------	-------------

--	--	--

Bold	Sample content for bold text .	For bold, add <code></code> or <code></code> to front and back of the text.
-------------	---------------------------------------	--

<i>Italic</i>	Sample content for <i>Italic text</i> .	For Italic, add <code><i></code> or <code></code> to front and back of the text.
---------------	---	--

<i>Bold and Italics</i>	Sample content for <i>bold and Italic text</i> .	For bold and Italics, add <code></code> or <code></code> to the front and back of the text.
--------------------------------	---	--

Heading 1	# Heading 1 content	For heading 1, add <code><h1></code> to start of the line.
-----------	---------------------	--

Heading 2	## Heading 2 content	For heading 2, add <code><h2></code> to start of the line.
-----------	----------------------	--

| Heading 3 | **###** Heading 3 content | For heading 3, add **###** to start of the line. |

| Heading 4 | **####** Heading 4 content | For heading 4, add **####** to start of the line. |

| Heading 5 | **#####** Heading 5 content | For heading 5, add **#####** to start of the line. |

| Heading 6 | **#####** Heading 6 content | For heading 6, add **#####** to start of the line. |

| Line Break | First line **
**Second line | For line break, press enter two times (or) add **
** in between the first and the second line. |

| Blockquotes | **>** Blockquotes text | For blockquotes, add **>** to start of the line. |

| Strike Through | Sample content for **~~strike through text~~**. | For strike through, add **~~** to front and back of the text. |

| Code (Single line) | **\Single line code** | For single line code, add **`** to front and back of the text. |

| Code block (Multi Line) | **\\\
Multi line code text
Multi line code text
** | For multiple line code, add **\\`** in the new line before and after the content. |

| Subscript | **_{**Subscript text**}** | For subscript, add **_{** to the front and **}** to the back of the text. |

| Superscript | **^{**Superscript text**}** | For superscript, add **^{** to the front and **}** to the back of the text. |

| Ordered List | **1.** First**
1.** Second | For ordered list, preceding one or more lines of text with **1.** |

| Unordered List | **First
second | For unordered list, preceding one or more lines of text with ***. |

| Links | **Link text without title text
`** Link text **`
Link text with title text
[** Link text **](URL , "title text")** | Create an inline link by wrapping link text in brackets **[]**, and then wrapping the URL as first parameter and title as second parameter in the parentheses **().****
Note:** The title text is optional, if needed it can be given manually. |

| Table | | Heading 1 | Heading 2 |**
|-----|-----|
| Col A1 | Col A2 |
**| Col B1 | Col B2 | | Create a table using the pipes and underscores as given in the syntax to create 2 x 2 table. |

| Horizontal Line | **(three asterix in new line)
(or)
(three underscores in new line) /**
For horizontal line, add or to the start of the new line. |

| Image | **** | Create an image by wrapping the image source in parentheses **().** |

| Image with alternate text | **![alternate text](URL path)** | Create an image with alternate text by wrapping an alternative text in brackets **[]**, and then link of the image source in parentheses **().****
Note:** When inserting the image using toolbar, the alternate text cannot be provided that needs to be given manually. |

| Escape tick marks supported | Sample text content with **bold and** not bold **text can be in the same line.** | In the syntax, the whole content is made as bold where the content not bold can be made as normal text by adding the bold syntax to the start and end of the respective text. Likewise you can do the same for various inline commands. |

| Escape Character | \ (any syntax) | Escape any markdown syntax by prefix \ to the syntax.
Example:
\Bold text

| HTML Entities | Copyright: © - ©
Trade mark: ™ - ™
Registered: ® - ®
Ampersand: & - &
Less than: < - <
Greater than: > - > | For HTML entities, add & and ; to the front and back of the respective entities. |

The above listed commands alone are supported in Syncfusion Markdown editor. For other unsupported commands, you can achieve using the HTML tags in Markdown editor. The foot notes, definitions, math, and check list markdown syntax are also not supported.

Markdown to HTML

The Rich Text Editor allows you to preview markdown changes immediately using preview. In this sample, the third-party library [Marked](#) is used to convert markdown into HTML content.

This sample demonstrates how to preview markdown changes in Rich Text Editor. Type or edit the display text, and apply format to view the preview of markdown. The [actionComplete](#) event can be used to convert Markdown to HTML.

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - Markdown to HTML Sample
 */
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { createElement } from '@syncfusion/ej2-base';
import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
class App extends React.Component {
  mdSource;
  mdSplit;
  htmlPreview;
  defaultRTE;
  textArea;
  value = `***Overview***
  The Rich Text Editor component is WYSIWYG ("what you see is what you get")
  editor used to create and edit the content and return valid HTML markup or
  markdown (MD) of the content. The editor provides a standard toolbar to
  format content using its commands. Modular library features to load the
  necessary functionality on demand. The toolbar contains commands to align the
  text, insert link, insert image, insert list, undo/redon operation, HTML view,
  and more.
  ***Key features***
  - *Mode*: Provides IFRAME and DIV mode.
  - *Module*: Modular library to load the necessary functionality on demand.
  - *Toolbar*: Provide a fully customizable toolbar.
  - *Editing*: HTML view to edit the source directly for developers.
  - *Third-party Integration*: Supports to integrate third-party library.
  - *Preview*: Preview the modified content before saving it.
  - *Tools*: Handling images, hyperlinks, video, uploads and more.`;
}
```

```

- *Undo and Redo*: Undo/redo manager.
- *Lists*: Creates bulleted and numbered list.`;
componentDidMount() {
  this.defaultRTE = new RichTextEditor({
    actionComplete: (e) => {
      if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args))
    {
      this.fullPreview({ mode: true, type: '' });
    }
    else if (!this.mdSplit.parentElement.classList.contains('e-
overlay')) {
      if (e.targetItem === 'Minimize') {
        this.textArea.style.display = 'block';
        this.textArea.style.width = '100%';
        if (this.htmlPreview) {
          this.htmlPreview.style.display = 'none';
        }
        this.mdSplit.classList.remove('e-active');
        this.mdSource.classList.remove('e-active');
      }
      this.markDownConversion();
    }
    setTimeout(() => {
      this.defaultRTE.toolbarModule.refreshToolbarOverflow();
    }, 400);
  },
  created: () => {
    this.textArea = this.defaultRTE.contentModule.getEditPanel();
    this.textArea.addEventListener('keyup', (e) => {
      this.markDownConversion();
    });
    this.mdSource = document.getElementById('preview-code');
    this.mdSource.addEventListener('click', (e) => {
      this.fullPreview({ mode: true, type: 'preview' });
      if (e.currentTarget.classList.contains('e-active')) {
        this.defaultRTE.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
          'Formats', 'OrderedList', 'UnorderedList', '|',
          'CreateLink', 'Image', 'Undo', 'Redo']);
      }
      e.currentTarget.parentElement.previousElementSibling.classList.add('e-
overlay');
    }
    else {
      this.defaultRTE.enableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
          'Formats', 'OrderedList', 'UnorderedList', '|',
          'CreateLink', 'Image', 'Undo', 'Redo']);
    }
    e.currentTarget.parentElement.previousElementSibling.classList.remove('e-
overlay');
  }
});
this.mdSplit = document.getElementById('MD_Preview');
this.mdSplit.addEventListener('click', (e) => {
  if (this.defaultRTE.element.classList.contains('e-rte-
full-screen')) {

```

```

        this.fullPreview({ mode: true, type: ' ' });
    }
    this.mdSource.classList.remove('e-active');
    if (!this.defaultRTE.element.classList.contains('e-rte-
full-screen')) {
        this.defaultRTE.showFullScreen();
    }
    });
    },
    editorMode: 'Markdown',
    height: '300px',
    toolbarSettings: {
        items: [
            'Bold', 'Italic', 'StrikeThrough', '|', 'Formats',
'OrderedList', 'UnorderedList', '|',
            'CreateLink', 'Image', '|',
            {
                template: '<button id="preview-code" class="e-tbar-
btn e-control e-btn e-icon-btn">' +
                    '<span class="e-btn-icon e-md-preview e-preview
e-icons"></span></button>',
                tooltipText: 'Preview'
            },
            {
                template: '<button id="MD_Preview" class="e-tbar-btn
e-control e-btn e-icon-btn">' +
                    '<span class="e-btn-icon e-view-side e-
icons"></span></button>',
                tooltipText: 'Split Editor',
            },
            'FullScreen', '|', 'Undo', 'Redo'
        ]
    },
    valueTemplate: this.value,
    });
    this.defaultRTE.appendTo('#markdownPreview');
}
markDownConversion() {
    if (this.mdSplit.classList.contains('e-active')) {
        const id = this.defaultRTE.getID() + 'html-preview';
        const htmlPreview = this.defaultRTE.element.querySelector('#' +
id);
        htmlPreview.innerHTML =
marked((this.defaultRTE.contentModule.getEditPanel()).value);
    }
}
fullPreview(e) {
    const id = this.defaultRTE.getID() + 'html-preview';
    this.htmlPreview = this.defaultRTE.element.querySelector('#' + id);
    if ((this.mdSource.classList.contains('e-active') ||
this.mdSplit.classList.contains('e-active')) && e.mode) {
        this.mdSource.classList.remove('e-active');
        this.mdSplit.classList.remove('e-active');
        this.mdSource.parentElement.title = 'Preview';
        this.textArea.style.display = 'block';
        this.textArea.style.width = '100%';
        this.htmlPreview.style.display = 'none';
    }
}

```

```

    }
    else {
      this.mdSource.classList.add('e-active');
      this.mdSplit.classList.add('e-active');
      if (!this.htmlPreview) {
        this.htmlPreview = createElement('div', { className: 'e-
content' });
        this.htmlPreview.id = id;
        this.textArea.parentNode.appendChild(this.htmlPreview);
      }
      if (e.type === 'preview') {
        this.textArea.style.display = 'none';
        this.htmlPreview.classList.add('e-pre-source');
        this.htmlPreview.style.width = '100%';
      }
      else {
        this.htmlPreview.classList.remove('e-pre-source');
        this.textArea.style.width = '50%';
        this.htmlPreview.style.width = '50%';
      }
      this.htmlPreview.style.display = 'block';
      this.htmlPreview.innerHTML =
marked((this.defaultRTE.contentModule.getEditPanel()).value);
      this.mdSource.parentElement.title = 'Code View';
    }
  }
  render() {
    return (<div id="rte-default" className='control-pane'>
    <div className='control-section' id="rtePreview">
      <div className="content-wrapper">
        <div id="markdownPreview"/>
      </div>
    </div>
    </div>);
  }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Markdown to HTML Sample
 */
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { createElement, KeyboardEventArgs } from '@syncfusion/ej2-base';
import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
class App extends React.Component<{}, {}> {
  public mdSource: any;
  public mdSplit: HTMLElement;
  public htmlPreview: any;
  public defaultRTE: RichTextEditor;
  public textArea: HTMLTextAreaElement;
  public value: string = `***Overview***

```

The Rich Text Editor component **is WYSIWYG** ("what you see is what you get") editor used to create and edit the content and **return** valid HTML markup or markdown (MD) of the content. The editor provides a standard toolbar to format content **using its** commands. Modular library features to load the necessary functionality on demand. The toolbar contains commands to align the text, insert link, insert image, insert list, undo/redo operation, HTML view, and more.

Key features

- ***Mode***: Provides IFRAME and DIV mode.
- ***Module***: Modular library to load the necessary functionality on demand.
- ***Toolbar***: Provide a fully customizable toolbar.
- ***Editing***: HTML view to edit the source directly **for** developers.
- ***Third-party Integration***: Supports to integrate third-party library.
- ***Preview***: Preview the modified content before saving it.
- ***Tools***: Handling images, hyperlinks, video, uploads and more.
- ***Undo and Redo***: Undo/redo manager.
- ***Lists***: Creates bulleted and numbered list.`;

```
public componentDidMount(): void {
  this.defaultRTE = new RichTextEditor({
    actionComplete: (e: any) => {
      if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args)) {
        this.fullPreview({ mode: true, type: '' });
      } else if (!(this.mdSplit as
any).parentElement.classList.contains('e-overlay')) {
        if (e.targetItem === 'Minimize') {
          this.textArea.style.display = 'block';
          this.textArea.style.width = '100%';
          if (this.htmlPreview) { this.htmlPreview.style.display = 'none';
        }

        this.mdSplit.classList.remove('e-active');
        this.mdSource.classList.remove('e-active');
      }
      this.markDownConversion();
    }
    setTimeout(() => {
      this.defaultRTE.toolbarModule.refreshToolbarOverflow();
    }, 400);
  },
  created: () => {
    this.textArea = (this.defaultRTE as
any).contentModule.getEditPanel();
    this.textArea.addEventListener('keyup', (e: KeyboardEventArgs) => {
      this.markDownConversion();
    });
    this.mdSource = document.getElementById('preview-code') as any;
    this.mdSource.addEventListener('click', (e: MouseEvent) => {
      this.fullPreview({ mode: true, type: 'preview' });
      if ((e.currentTarget as HTMLElement).classList.contains('e-active'))
    {
      this.defaultRTE.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
      'Formats', 'OrderedList', 'UnorderedList', '|',
      'CreateLink', 'Image', 'Undo', 'Redo']);
      (e.currentTarget as
any).parentElement.previousElementSibling.classList.add('e-overlay');
    } else {
```

```

        this.defaultRTE.enableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
        'Formats', 'OrderedList', 'UnorderedList', '|',
        'CreateLink', 'Image', 'Undo', 'Redo']);
        (e.currentTarget as
any).parentElement.previousElementSibling.classList.remove('e-overlay');
    }
    });
    this.mdSplit = document.getElementById('MD_Preview') as any;
    this.mdSplit.addEventListener('click', (e: MouseEvent) => {
        if (this.defaultRTE.element.classList.contains('e-rte-full-screen'))
    {
        this.fullPreview({ mode: true, type: '' });
    }
    this.mdSource.classList.remove('e-active');
    if (!this.defaultRTE.element.classList.contains('e-rte-full-screen'))
    {
        this.defaultRTE.showFullScreen();
    }
    });
    },
    editorMode: 'Markdown',
    height: '300px',
    toolbarSettings: {
        items: [
            'Bold', 'Italic', 'StrikeThrough', '|', 'Formats', 'OrderedList',
'UnorderedList', '|',
            'CreateLink', 'Image', '|',
            {
                template: '<button id="preview-code" class="e-tbar-btn e-control
e-btn e-icon-btn">' +
                    '<span class="e-btn-icon e-md-preview e-preview e-
icons"></span></button>',
                tooltipText: 'Preview'
            },
            {
                template: '<button id="MD_Preview" class="e-tbar-btn e-control e-
btn e-icon-btn">' +
                    '<span class="e-btn-icon e-view-side e-
icons"></span></button>',
                tooltipText: 'Split Editor',
            },
            'FullScreen', '|', 'Undo', 'Redo']
        },
        valueTemplate: this.value,
    });
    this.defaultRTE.appendTo('#markdownPreview');
}

public markdownConversion(): void {
    if (this.mdSplit.classList.contains('e-active')) {
        const id: string = this.defaultRTE.getID() + 'html-preview';
        const htmlPreview: any = this.defaultRTE.element.querySelector('#' +
id);
        htmlPreview.innerHTML = marked(((this as
any).defaultRTE.contentModule.getEditPanel()).value);
    }
}

```

```

public fullPreview(e: { [key: string]: string | boolean }): void {
    const id: string = this.defaultRTE.getID() + 'html-preview';
    this.htmlPreview = this.defaultRTE.element.querySelector('#' + id);
    if ((this.mdSource.classList.contains('e-active') ||
    this.mdSplit.classList.contains('e-active')) && e.mode) {
        this.mdSource.classList.remove('e-active');
        this.mdSplit.classList.remove('e-active');
        this.mdSource.parentElement.title = 'Preview';
        this.textArea.style.display = 'block';
        this.textArea.style.width = '100%';
        this.htmlPreview.style.display = 'none';
    } else {
        this.mdSource.classList.add('e-active');
        this.mdSplit.classList.add('e-active');
        if (!this.htmlPreview) {
            this.htmlPreview = createElement('div', { className: 'e-content' });
            this.htmlPreview.id = id;
            (this.textArea as any).parentNode.appendChild(this.htmlPreview);
        }
        if (e.type === 'preview') {
            this.textArea.style.display = 'none';
            this.htmlPreview.classList.add('e-pre-source');
            this.htmlPreview.style.width = '100%';
        } else {
            this.htmlPreview.classList.remove('e-pre-source');
            this.textArea.style.width = '50%';
            this.htmlPreview.style.width = '50%';
        }
        this.htmlPreview.style.display = 'block';
        this.htmlPreview.innerHTML = marked((this as
any).defaultRTE.contentModule.getEditPanel()).value);
        this.mdSource.parentElement.title = 'Code View';
    }
}

public render() {
    return (
        <div id="rte-default" className='control-pane'>
            <div className='control-section' id="rtePreview">
                <div className="content-wrapper">
                    <div id="markdownPreview"/>
                </div>
            </div>
        </div>
    );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Markdown to HTML Sample
 */
import { isNullOrUndefined } from '@syncfusion/ej2-base';

```

```

import { createElement } from '@syncfusion/ej2-base';
import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
import { useEffect } from "react";
function App() {
    useEffect(() => {
        defaultRTE = new RichTextEditor({
            actionComplete: (e) => {
                if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args))
                {
                    fullPreview({ mode: true, type: '' });
                }
                else if (!mdSplit.parentElement.classList.contains('e-overlay')) {
                    if (e.targetItem === 'Minimize') {
                        textArea.style.display = 'block';
                        textArea.style.width = '100%';
                        if (htmlPreview) {
                            htmlPreview.style.display = 'none';
                        }
                        mdSplit.classList.remove('e-active');
                        mdSource.classList.remove('e-active');
                    }
                    markDownConversion();
                }
                setTimeout(() => {
                    defaultRTE.toolbarModule.refreshToolbarOverflow();
                }, 400);
            },
            created: () => {
                textArea = defaultRTE.contentModule.getEditPanel();
                textArea.addEventListener('keyup', (e) => {
                    markDownConversion();
                });
                mdSource = document.getElementById('preview-code');
                mdSource.addEventListener('click', (e) => {
                    fullPreview({ mode: true, type: 'preview' });
                    if (e.currentTarget.classList.contains('e-active')) {
                        defaultRTE.disableToolbarItem(['Bold', 'Italic', 'StrikeThrough', '|', 'Formats', 'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', 'Undo', 'Redo']);
                    }
                    e.currentTarget.parentElement.previousElementSibling.classList.add('e-overlay');
                }
                else {
                    defaultRTE.enableToolbarItem(['Bold', 'Italic', 'StrikeThrough', '|', 'Formats', 'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', 'Undo', 'Redo']);
                    e.currentTarget.parentElement.previousElementSibling.classList.remove('e-overlay');
                }
            }
        });
    });
}

```



```

    });
    mdSplit = document.getElementById('MD_Preview');
    mdSplit.addEventListener('click', (e) => {
      if (defaultRTE.element.classList.contains('e-rte-full-
screen')) {
        fullPreview({ mode: true, type: '' });
      }
      mdSource.classList.remove('e-active');
      if (!defaultRTE.element.classList.contains('e-rte-full-
screen')) {
        defaultRTE.showFullScreen();
      }
    });
  },
  editorMode: 'Markdown',
  height: '300px',
  toolbarSettings: {
    items: [
      'Bold', 'Italic', 'StrikeThrough', '|', 'Formats',
      'OrderedList', 'UnorderedList', '|',
      'CreateLink', 'Image', '|',
      {
        template: '<button id="preview-code" class="e-tbar-
btn e-control e-btn e-icon-btn">' +
          '<span class="e-btn-icon e-md-preview e-preview
e-icons"></span></button>',
        tooltipText: 'Preview'
      },
      {
        template: '<button id="MD_Preview" class="e-tbar-btn
e-control e-btn e-icon-btn">' +
          '<span class="e-btn-icon e-view-side e-
icons"></span></button>',
        tooltipText: 'Split Editor',
      },
      'FullScreen', '|', 'Undo', 'Redo'
    ]
  },
  valueTemplate: value,
});
defaultRTE.appendTo('#markdownPreview');
});
let mdSource;
let mdSplit;
let htmlPreview;
let defaultRTE;
let textArea;
let value = `***Overview***

```

The Rich Text Editor component is **WYSIWYG** ("what you see is what you get") editor used to create and edit the content and **return** valid HTML markup or markdown (MD) of the content. The editor provides a standard toolbar to format content **using its** commands. Modular library features to load the necessary functionality on demand. The toolbar contains commands to align the text, insert link, insert image, insert list, undo/redo operation, HTML view, and more.

Key features

- *Mode*: Provides IFRAME and DIV mode.

```

- *Module*: Modular library to load the necessary functionality on demand.
- *Toolbar*: Provide a fully customizable toolbar.
- *Editing*: HTML view to edit the source directly for developers.
- *Third-party Integration*: Supports to integrate third-party library.
- *Preview*: Preview the modified content before saving it.
- *Tools*: Handling images, hyperlinks, video, uploads and more.
- *Undo and Redo*: Undo/redo manager.
- *Lists*: Creates bulleted and numbered list.`;
function markDownConversion() {
    if (mdSplit.classList.contains('e-active')) {
        const id = defaultRTE.getID() + 'html-preview';
        const htmlPreview = defaultRTE.element.querySelector('#' + id);
        htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
    }
}
function fullPreview(e) {
    const id = defaultRTE.getID() + 'html-preview';
    htmlPreview = defaultRTE.element.querySelector('#' + id);
    if ((mdSource.classList.contains('e-active') ||
mdSplit.classList.contains('e-active')) && e.mode) {
        mdSource.classList.remove('e-active');
        mdSplit.classList.remove('e-active');
        mdSource.parentElement.title = 'Preview';
        textArea.style.display = 'block';
        textArea.style.width = '100%';
        htmlPreview.style.display = 'none';
    }
    else {
        mdSource.classList.add('e-active');
        mdSplit.classList.add('e-active');
        if (!htmlPreview) {
            htmlPreview = createElement('div', { className: 'e-content'
});

            htmlPreview.id = id;
            textArea.parentNode.appendChild(htmlPreview);
        }
        if (e.type === 'preview') {
            textArea.style.display = 'none';
            htmlPreview.classList.add('e-pre-source');
            htmlPreview.style.width = '100%';
        }
        else {
            htmlPreview.classList.remove('e-pre-source');
            textArea.style.width = '50%';
            htmlPreview.style.width = '50%';
        }
        htmlPreview.style.display = 'block';
        htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
        mdSource.parentElement.title = 'Code View';
    }
}
return (<div id="rte-default" className='control-pane'>
    <div className='control-section' id="rtePreview">
        <div className="content-wrapper">
            <div id="markdownPreview"/>

```

```

        </div>
      </div>
    </div>);
  }
  export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Markdown to HTML Sample
 */
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { createElement, KeyboardEventArgs } from '@syncfusion/ej2-base';
import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
import { useEffect } from 'react';
function App() {
  useEffect(() => {
    defaultRTE = new RichTextEditor({
      actionComplete: (e: any) => {
        if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args)) {
          fullPreview({ mode: true, type: '' });
        } else if (!(mdSplit as any).parentElement.classList.contains('e-overlay')) {
          if (e.targetItem === 'Minimize') {
            textArea.style.display = 'block';
            textArea.style.width = '100%';
            if (htmlPreview) { htmlPreview.style.display = 'none'; }
            mdSplit.classList.remove('e-active');
            mdSource.classList.remove('e-active');
          }
          markDownConversion();
        }
        setTimeout(() => {
          defaultRTE.toolbarModule.refreshToolbarOverflow();
        }, 400);
      },
      created: () => {
        textArea = (defaultRTE as any).contentModule.getEditPanel();
        textArea.addEventListener('keyup', (e: KeyboardEventArgs) => {
          markDownConversion();
        });
        mdSource = document.getElementById('preview-code') as any;
        mdSource.addEventListener('click', (e: MouseEvent) => {
          fullPreview({ mode: true, type: 'preview' });
          if ((e.currentTarget as HTMLElement).classList.contains('e-active'))
          {
            defaultRTE.disableToolbarItem(['Bold', 'Italic', 'StrikeThrough',
            '|',
            'Formats', 'OrderedList', 'UnorderedList', '|',
            'CreateLink', 'Image', 'Undo', 'Redo']);
            (e.currentTarget as any).parentElement.previousElementSibling.classList.add('e-overlay');
          } else {

```

```

        defaultRTE.enableToolbarItem(['Bold', 'Italic', 'StrikeThrough',
        '|',
        'Formats', 'OrderedList', 'UnorderedList', '|',
        'CreateLink', 'Image', 'Undo', 'Redo']);
        (e.currentTarget as
any).parentElement.previousElementSibling.classList.remove('e-overlay');
    }
    });
    mdSplit = document.getElementById('MD_Preview') as any;
    mdSplit.addEventListener('click', (e: MouseEvent) => {
        if (defaultRTE.element.classList.contains('e-rte-full-screen')) {
            fullPreview({ mode: true, type: '' });
        }
        mdSource.classList.remove('e-active');
        if (!defaultRTE.element.classList.contains('e-rte-full-screen')) {
            defaultRTE.showFullScreen();
        }
    });
    },
    editorMode: 'Markdown',
    height: '300px',
    toolbarSettings: {
        items: [
            'Bold', 'Italic', 'StrikeThrough', '|', 'Formats', 'OrderedList',
            'UnorderedList', '|',
            'CreateLink', 'Image', '|',
            {
                template: '<button id="preview-code" class="e-tbar-btn e-control
e-btn e-icon-btn">' +
                '<span class="e-btn-icon e-md-preview e-preview e-
icons"></span></button>',
                tooltipText: 'Preview'
            },
            {
                template: '<button id="MD_Preview" class="e-tbar-btn e-control e-
btn e-icon-btn">' +
                '<span class="e-btn-icon e-view-side e-
icons"></span></button>',
                tooltipText: 'Split Editor',
            },
            'FullScreen', '|', 'Undo', 'Redo']
        },
        valueTemplate: value,
    });
    defaultRTE.appendTo('#markdownPreview');
    });
    let mdSource: any;
    let mdSplit: HTMLElement;
    let htmlPreview: any;
    let defaultRTE: RichTextEditor;
    let textArea: HTMLTextAreaElement;
    let value: string = `***Overview***`
    The Rich Text Editor component is WYSIWYG ("what you see is what you get")
    editor used to create and edit the content and return valid HTML markup or
    markdown (MD) of the content. The editor provides a standard toolbar to
    format content using its commands. Modular library features to load the
    necessary functionality on demand. The toolbar contains commands to align the

```

text, insert link, insert image, insert list, undo/redo operation, HTML view, and more.

Key features

- *Mode*: Provides IFRAME and DIV mode.
- *Module*: Modular library to load the necessary functionality on demand.
- *Toolbar*: Provide a fully customizable toolbar.
- *Editing*: HTML view to edit the source directly **for** developers.
- *Third-party Integration*: Supports to integrate third-party library.
- *Preview*: Preview the modified content before saving it.
- *Tools*: Handling images, hyperlinks, video, uploads and more.
- *Undo and Redo*: Undo/redo manager.
- *Lists*: Creates bulleted and numbered list.`;

```
function markdownConversion(): void {
  if (mdSplit.classList.contains('e-active')) {
    const id: string = defaultRTE.getID() + 'html-preview';
    const htmlPreview: any = defaultRTE.element.querySelector('#' + id);
    htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
  }
}

function fullPreview(e: { [key: string]: string | boolean }): void {
  const id: string = defaultRTE.getID() + 'html-preview';
  htmlPreview = defaultRTE.element.querySelector('#' + id);
  if ((mdSource.classList.contains('e-active') ||
mdSplit.classList.contains('e-active')) && e.mode) {
    mdSource.classList.remove('e-active');
    mdSplit.classList.remove('e-active');
    mdSource.parentElement.title = 'Preview';
    textArea.style.display = 'block';
    textArea.style.width = '100%';
    htmlPreview.style.display = 'none';
  } else {
    mdSource.classList.add('e-active');
    mdSplit.classList.add('e-active');
    if (!htmlPreview) {
      htmlPreview = createElement('div', { className: 'e-content' });
      htmlPreview.id = id;
      (textArea as any).parentNode.appendChild(htmlPreview);
    }
    if (e.type === 'preview') {
      textArea.style.display = 'none'; htmlPreview.classList.add('e-pre-
source');
      htmlPreview.style.width = '100%';
    } else {
      htmlPreview.classList.remove('e-pre-source');
      textArea.style.width = '50%';
      htmlPreview.style.width = '50%';
    }
    htmlPreview.style.display = 'block';
    htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
    mdSource.parentElement.title = 'Code View';
  }
}

return (
  <div id="rte-default" className='control-pane'>
    <div className='control-section' id="rtePreview">
```

```

        <div className="content-wrapper">
          <div id="markdownPreview"/>
        </div>
      </div>
    </div>
  );
}
export default App;

```

Table

Rich Text Editor allows to insert Markdown table in edit panel with 2 X 2 rows and columns along with the heading.

To use table tool, add the **CreateTable** item in toolbar items.

Insert table

To insert the table in Rich Text Editor, click the **table** toolbar option to insert the table into Rich Text Editor content and this is the default way in all the devices.

Please refer the below sample and code snippets to add the table in Markdown editor

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - Markdown to HTML Sample
 */
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { createElement } from '@syncfusion/ej2-base';
import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
class App extends React.Component {
  mdSource;
  mdSplit;
  htmlPreview;
  defaultRTE;
  textArea;
  value = `***Overview***
  The Rich Text Editor component is WYSIWYG ("what you see is what you get")
  editor used to create and edit the content and return valid HTML markup or
  markdown (MD) of the content. The editor provides a standard toolbar to
  format content using its commands. Modular library features to load the
  necessary functionality on demand. The toolbar contains commands to align the
  text, insert link, insert image, insert list, undo/redo operation, HTML view,
  and more.
  ***Key features***
  - *Mode*: Provides IFRAME and DIV mode.
  - *Module*: Modular library to load the necessary functionality on demand.
  - *Toolbar*: Provide a fully customizable toolbar.
  - *Editing*: HTML view to edit the source directly for developers.
  - *Third-party Integration*: Supports to integrate third-party library.
  - *Preview*: Preview the modified content before saving it.
  - *Tools*: Handling images, hyperlinks, video, uploads and more.

```

```

- *Undo and Redo*: Undo/redo manager.
- *Lists*: Creates bulleted and numbered list.`;
componentDidMount() {
  this.defaultRTE = new RichTextEditor({
    actionComplete: (e) => {
      if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args))
{
      this.fullPreview({ mode: true, type: '' });
    }
    else if (!this.mdSplit.parentElement.classList.contains('e-
overlay')) {
      if (e.targetItem === 'Minimize') {
        this.textArea.style.display = 'block';
        this.textArea.style.width = '100%';
        if (this.htmlPreview) {
          this.htmlPreview.style.display = 'none';
        }
        this.mdSplit.classList.remove('e-active');
        this.mdSource.classList.remove('e-active');
      }
      this.markDownConversion();
    }
    setTimeout(() => {
      this.defaultRTE.toolbarModule.refreshToolbarOverflow();
    }, 400);
  },
  created: () => {
    this.textArea = this.defaultRTE.contentModule.getEditPanel();
    this.textArea.addEventListener('keyup', (e) => {
      this.markDownConversion();
    });
    this.mdSource = document.getElementById('preview-code');
    this.mdSource.addEventListener('click', (e) => {
      this.fullPreview({ mode: true, type: 'preview' });
      if (e.currentTarget.classList.contains('e-active')) {
        this.defaultRTE.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
          'Formats', 'OrderedList', 'UnorderedList', '|',
          'CreateLink', 'Image', 'Undo', 'Redo',
'StrikeThrough', '|',
          'Formats', 'OrderedList', 'UnorderedList', '|',
          'CreateLink', 'Image', 'Undo', 'Redo',
'StrikeThrough', '|',
          'Formats', 'OrderedList', 'UnorderedList', '|',
          'CreateLink', 'Image', 'Undo', 'Redo',
          'CreateTable']);
        e.currentTarget.parentElement.previousElementSibling.classList.add('e-
overlay');
      }
      else {
        this.defaultRTE.enableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
          'Formats', 'OrderedList', 'UnorderedList', '|',
          'CreateLink', 'Image', 'Undo', 'Redo',
          'CreateTable']);
        e.currentTarget.parentElement.previousElementSibling.classList.remove('e-
overlay');
      }
    });
    this.mdSplit = document.getElementById('MD_Preview');
    this.mdSplit.addEventListener('click', (e) => {

```

```

        if (this.defaultRTE.element.classList.contains('e-rte-
full-screen')) {
            this.fullPreview({ mode: true, type: '' });
        }
        this.mdSource.classList.remove('e-active');
        if (!this.defaultRTE.element.classList.contains('e-rte-
full-screen')) {
            this.defaultRTE.showFullScreen();
        }
    });
},
editorMode: 'Markdown',
height: '300px',
toolbarSettings: {
    items: [
        'Bold', 'Italic', 'StrikeThrough', '|', 'Formats',
'OrderedList', 'UnorderedList', '|',
        'CreateLink', 'Image', 'CreateTable', '|',
        {
            template: '<button id="preview-code" class="e-tbar-
btn e-control e-btn e-icon-btn">' +
                '<span class="e-btn-icon e-md-preview e-preview
e-icons"></span></button>',
            tooltipText: 'Preview'
        },
        {
            template: '<button id="MD_Preview" class="e-tbar-btn
e-control e-btn e-icon-btn">' +
                '<span class="e-btn-icon e-view-side e-
icons"></span></button>',
            tooltipText: 'Split Editor'
        },
        'FullScreen', '|', 'Undo', 'Redo'
    ]
},
valueTemplate: this.value
});
this.defaultRTE.appendTo('#markdownPreview');
}
markDownConversion() {
    if (this.mdSplit.classList.contains('e-active')) {
        const id = this.defaultRTE.getID() + 'html-preview';
        const htmlPreview = this.defaultRTE.element.querySelector('#' +
id);
        htmlPreview.innerHTML =
marked((this.defaultRTE.contentModule.getEditPanel()).value);
    }
}
fullPreview(e) {
    const id = this.defaultRTE.getID() + 'html-preview';
    this.htmlPreview = this.defaultRTE.element.querySelector('#' + id);
    if ((this.mdSource.classList.contains('e-active') ||
this.mdSplit.classList.contains('e-active')) && e.mode) {
        this.mdSource.classList.remove('e-active');
        this.mdSplit.classList.remove('e-active');
        this.mdSource.parentElement.title = 'Preview';
        this.textArea.style.display = 'block';
    }
}

```



```

        this.textArea.style.width = '100%';
        this.htmlPreview.style.display = 'none';
    }
    else {
        this.mdSource.classList.add('e-active');
        this.mdSplit.classList.add('e-active');
        if (!this.htmlPreview) {
            this.htmlPreview = createElement('div', { className: 'e-
content' });
            this.htmlPreview.id = id;
            this.textArea.parentNode.appendChild(this.htmlPreview);
        }
        if (e.type === 'preview') {
            this.textArea.style.display = 'none';
            this.htmlPreview.classList.add('e-pre-source');
            this.htmlPreview.style.width = '100%';
        }
        else {
            this.htmlPreview.classList.remove('e-pre-source');
            this.textArea.style.width = '50%';
            this.htmlPreview.style.width = '50%';
        }
        this.htmlPreview.style.display = 'block';
        this.htmlPreview.innerHTML =
marked((this.defaultRTE.contentModule.getEditPanel()).value);
        this.mdSource.parentElement.title = 'Code View';
    }
}
render() {
    return (<div id="rte-default" className='control-pane'>
    <div className='control-section' id="rtePreview">
        <div className="content-wrapper">
            <div id="markdownPreview"/>
        </div>
    </div>
    </div>);
}
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Markdown to HTML Sample
 */
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { createElement, KeyboardEventArgs } from '@syncfusion/ej2-base';
import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
class App extends React.Component<{}, {}> {
    public mdSource: HTMLElement;
    public mdSplit: HTMLElement;
    public htmlPreview: HTMLElement;
    public defaultRTE: RichTextEditor;
}

```

```

public textArea: HTMLTextAreaElement;
public value: string = `***Overview***`
The Rich Text Editor component is WYSIWYG ("what you see is what you get")
editor used to create and edit the content and return valid HTML markup or
markdown (MD) of the content. The editor provides a standard toolbar to
format content using its commands. Modular library features to load the
necessary functionality on demand. The toolbar contains commands to align the
text, insert link, insert image, insert list, undo/redo operation, HTML view,
and more.
***Key features***
- *Mode*: Provides IFRAME and DIV mode.
- *Module*: Modular library to load the necessary functionality on demand.
- *Toolbar*: Provide a fully customizable toolbar.
- *Editing*: HTML view to edit the source directly for developers.
- *Third-party Integration*: Supports to integrate third-party library.
- *Preview*: Preview the modified content before saving it.
- *Tools*: Handling images, hyperlinks, video, uploads and more.
- *Undo and Redo*: Undo/redo manager.
- *Lists*: Creates bulleted and numbered list.`;
public componentDidMount(): void {
  this.defaultRTE = new RichTextEditor({
    actionComplete: (e: any) => {
      if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args)) {
        this.fullPreview({ mode: true, type: ' ' });
      } else if (!(this as
any).mdSplit.parentElement.classList.contains('e-overlay')) {
        if (e.targetItem === 'Minimize') {
          this.textArea.style.display = 'block';
          this.textArea.style.width = '100%';
          if (this.htmlPreview) { this.htmlPreview.style.display = 'none';
}

          this.mdSplit.classList.remove('e-active');
          this.mdSource.classList.remove('e-active');
        }
        this.markDownConversion();
      }
      setTimeout(() => {
        this.defaultRTE.toolbarModule.refreshToolbarOverflow();
      }, 400);
    },
    created: () => {
      this.textArea = (this.defaultRTE as
any).contentModule.getEditPanel();
      this.textArea.addEventListener('keyup', (e: KeyboardEventArgs) => {
        this.markDownConversion();
      });
      this.mdSource = document.getElementById('preview-code') as any;
      this.mdSource.addEventListener('click', (e: MouseEvent) => {
        this.fullPreview({ mode: true, type: 'preview' });
        if ((e.currentTarget as HTMLElement).classList.contains('e-active'))
{
          this.defaultRTE.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
'Formats', 'OrderedList', 'UnorderedList', '|',
'CreateLink', 'Image', 'Undo', 'Redo', 'CreateTable']);
          (e.currentTarget as
any).parentElement.previousElementSibling.classList.add('e-overlay');

```

```

    } else {
        this.defaultRTE.enableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
        'Formats', 'OrderedList', 'UnorderedList', '|',
        'CreateLink', 'Image', 'Undo', 'Redo', 'CreateTable']);
        (e.currentTarget as
any).parentElement.previousElementSibling.classList.remove('e-overlay');
    }
});
this.mdSplit = document.getElementById('MD_Preview') as any;
this.mdSplit.addEventListener('click', (e: MouseEvent) => {
    if (this.defaultRTE.element.classList.contains('e-rte-full-screen'))
{
        this.fullPreview({ mode: true, type: '' });
    }
    this.mdSource.classList.remove('e-active');
    if (!this.defaultRTE.element.classList.contains('e-rte-full-screen'))
{
        this.defaultRTE.showFullScreen();
    }
});
},
editorMode: 'Markdown',
height: '300px',
toolbarSettings: {
    items: [
        'Bold', 'Italic', 'StrikeThrough', '|', 'Formats', 'OrderedList',
'UnorderedList', '|',
        'CreateLink', 'Image', 'CreateTable', '|',
        {
            template: '<button id="preview-code" class="e-tbar-btn e-control
e-btn e-icon-btn">' +
                '<span class="e-btn-icon e-md-preview e-preview e-
icons"></span></button>',
            tooltipText: 'Preview'
        },
        {
            template: '<button id="MD_Preview" class="e-tbar-btn e-control e-
btn e-icon-btn">' +
                '<span class="e-btn-icon e-view-side e-
icons"></span></button>',
            tooltipText: 'Split Editor'
        },
        'FullScreen', '|', 'Undo', 'Redo'
    ],
    valueTemplate: this.value
});
this.defaultRTE.appendTo('#markdownPreview');
}
public markDownConversion(): void {
    if (this.mdSplit.classList.contains('e-active')) {
        const id: string = this.defaultRTE.getID() + 'html-preview';
        const htmlPreview: any = this.defaultRTE.element.querySelector('#' +
id);
        htmlPreview.innerHTML = marked(((this.defaultRTE as
any).contentModule.getEditPanel()).value);
    }
}

```

```

    }
    public fullPreview(e: { [key: string]: string | boolean }): void {
        const id: string = this.defaultRTE.getID() + 'html-preview';
        this.htmlPreview = this.defaultRTE.element.querySelector('#' + id);
        if ((this.mdSource.classList.contains('e-active') ||
        this.mdSplit.classList.contains('e-active')) && e.mode) {
            this.mdSource.classList.remove('e-active');
            this.mdSplit.classList.remove('e-active');
            this.mdSource.parentElement.title = 'Preview';
            this.textArea.style.display = 'block';
            this.textArea.style.width = '100%';
            this.htmlPreview.style.display = 'none';
        } else {
            this.mdSource.classList.add('e-active');
            this.mdSplit.classList.add('e-active');
            if (!this.htmlPreview) {
                this.htmlPreview = createElement('div', { className: 'e-content' });
                this.htmlPreview.id = id;
                (this.textArea as any).parentNode.appendChild(this.htmlPreview);
            }
            if (e.type === 'preview') {
                this.textArea.style.display = 'none';
                this.htmlPreview.classList.add('e-pre-source');
                this.htmlPreview.style.width = '100%';
            } else {
                this.htmlPreview.classList.remove('e-pre-source');
                this.textArea.style.width = '50%';
                this.htmlPreview.style.width = '50%';
            }
            this.htmlPreview.style.display = 'block';
            this.htmlPreview.innerHTML = marked(((this as
any).defaultRTE.contentModule.getEditPanel()).value);
            this.mdSource.parentElement.title = 'Code View';
        }
    }
    public render() {
        return (
            <div id="rte-default" className='control-pane'>
                <div className='control-section' id="rtePreview">
                    <div className="content-wrapper">
                        <div id="markdownPreview"/>
                    </div>
                </div>
            </div>
        );
    }
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Markdown to HTML Sample
 */

```

```

import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { createElement } from '@syncfusion/ej2-base';
import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
import { useEffect } from "react";
function App() {
    useEffect(() => {
        defaultRTE = new RichTextEditor({
            actionComplete: (e) => {
                if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args))
                {
                    fullPreview({ mode: true, type: '' });
                }
                else if (!this.mdSplit.parentElement.classList.contains('e-overlay')) {
                    if (e.targetItem === 'Minimize') {
                        textArea.style.display = 'block';
                        textArea.style.width = '100%';
                        if (htmlPreview) {
                            htmlPreview.style.display = 'none';
                        }
                        mdSplit.classList.remove('e-active');
                        mdSource.classList.remove('e-active');
                    }
                    markDownConversion();
                }
                setTimeout(() => {
                    defaultRTE.toolbarModule.refreshToolbarOverflow();
                }, 400);
            },
            created: () => {
                textArea = defaultRTE.contentModule.getEditPanel();
                textArea.addEventListener('keyup', (e) => {
                    markDownConversion();
                });
                mdSource = document.getElementById('preview-code');
                mdSource.addEventListener('click', (e) => {
                    fullPreview({ mode: true, type: 'preview' });
                    if (e.currentTarget.classList.contains('e-active')) {
                        defaultRTE.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
'Formats', 'OrderedList', 'UnorderedList', '|',
'CreateLink', 'Image', 'Undo', 'Redo',
'CreateTable']);
e.currentTarget.parentElement.previousElementSibling.classList.add('e-overlay');
                    }
                    else {
                        defaultRTE.enableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
'Formats', 'OrderedList', 'UnorderedList', '|',
'CreateLink', 'Image', 'Undo', 'Redo',
'CreateTable']);

```

```

e.currentTarget.parentElement.previousElementSibling.classList.remove('e-
overlay');
    }
  });
  mdSplit = document.getElementById('MD_Preview');
  mdSplit.addEventListener('click', (e) => {
    if (defaultRTE.element.classList.contains('e-rte-full-
screen')) {
      fullPreview({ mode: true, type: '' });
    }
    mdSource.classList.remove('e-active');
    if (!defaultRTE.element.classList.contains('e-rte-full-
screen')) {
      defaultRTE.showFullScreen();
    }
  });
},
editorMode: 'Markdown',
height: '300px',
toolbarSettings: {
  items: [
    'Bold', 'Italic', 'StrikeThrough', '|', 'Formats',
    'OrderedList', 'UnorderedList', '|',
    'CreateLink', 'Image', 'CreateTable', '|',
    {
      template: '<button id="preview-code" class="e-tbar-
btn e-control e-btn e-icon-btn">' +
        '<span class="e-btn-icon e-md-preview e-preview
e-icons"></span></button>',
      tooltipText: 'Preview'
    },
    {
      template: '<button id="MD_Preview" class="e-tbar-btn
e-control e-btn e-icon-btn">' +
        '<span class="e-btn-icon e-view-side e-
icons"></span></button>',
      tooltipText: 'Split Editor'
    },
    'FullScreen', '|', 'Undo', 'Redo'
  ]
},
valueTemplate: value
});
defaultRTE.appendTo('#markdownPreview');
});
let mdSource;
let mdSplit;
let htmlPreview;
let defaultRTE;
let textArea;
let value = `***Overview***

```

The Rich Text Editor component **is WYSIWYG** ("what you see is what you get") editor used to create and edit the content and **return** valid HTML markup or markdown (MD) of the content. The editor provides a standard toolbar to format content **using its** commands. Modular library features to load the necessary functionality on demand. The toolbar contains commands to align the

```

text, insert link, insert image, insert list, undo/redo operation, HTML view,
and more.
***Key features***
- *Mode*: Provides IFRAME and DIV mode.
- *Module*: Modular library to load the necessary functionality on demand.
- *Toolbar*: Provide a fully customizable toolbar.
- *Editing*: HTML view to edit the source directly for developers.
- *Third-party Integration*: Supports to integrate third-party library.
- *Preview*: Preview the modified content before saving it.
- *Tools*: Handling images, hyperlinks, video, uploads and more.
- *Undo and Redo*: Undo/redo manager.
- *Lists*: Creates bulleted and numbered list.`;
function markdownConversion() {
    if (mdSplit.classList.contains('e-active')) {
        const id = defaultRTE.getID() + 'html-preview';
        const htmlPreview = defaultRTE.element.querySelector('#' + id);
        htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
    }
}
function fullPreview(e) {
    const id = defaultRTE.getID() + 'html-preview';
    htmlPreview = defaultRTE.element.querySelector('#' + id);
    if ((mdSource.classList.contains('e-active') ||
mdSplit.classList.contains('e-active')) && e.mode) {
        mdSource.classList.remove('e-active');
        mdSplit.classList.remove('e-active');
        mdSource.parentElement.title = 'Preview';
        textArea.style.display = 'block';
        textArea.style.width = '100%';
        htmlPreview.style.display = 'none';
    }
    else {
        mdSource.classList.add('e-active');
        mdSplit.classList.add('e-active');
        if (!htmlPreview) {
            htmlPreview = createElement('div', { className: 'e-content'
});

            htmlPreview.id = id;
            textArea.parentNode.appendChild(htmlPreview);
        }
        if (e.type === 'preview') {
            textArea.style.display = 'none';
            htmlPreview.classList.add('e-pre-source');
            htmlPreview.style.width = '100%';
        }
        else {
            htmlPreview.classList.remove('e-pre-source');
            textArea.style.width = '50%';
            htmlPreview.style.width = '50%';
        }
        htmlPreview.style.display = 'block';
        htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
        mdSource.parentElement.title = 'Code View';
    }
}
}

```

```

    return (<div id="rte-default" className='control-pane'>
      <div className='control-section' id="rtePreview">
        <div className="content-wrapper">
          <div id="markdownPreview"/>
        </div>
      </div>
    </div>);
  }
  export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Markdown to HTML Sample
 */
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { createElement, KeyboardEventArgs } from '@syncfusion/ej2-base';
import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
import { useEffect } from 'react';
function App() {
  useEffect(() => {
    defaultRTE = new RichTextEditor({
      actionComplete: (e: any) => {
        if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args)) {
          fullPreview({ mode: true, type: '' });
        } else if (!(this as any).mdSplit.parentElement.classList.contains('e-overlay')) {
          if (e.targetItem === 'Minimize') {
            textArea.style.display = 'block';
            textArea.style.width = '100%';
            if (htmlPreview) { htmlPreview.style.display = 'none'; }
            mdSplit.classList.remove('e-active');
            mdSource.classList.remove('e-active');
          }
          markDownConversion();
        }
        setTimeout(() => {
          defaultRTE.toolbarModule.refreshToolbarOverflow();
        }, 400);
      },
      created: () => {
        textArea = (defaultRTE as any).contentModule.getEditPanel();
        textArea.addEventListener('keyup', (e: KeyboardEventArgs) => {
          markDownConversion();
        });
        mdSource = document.getElementById('preview-code') as any;
        mdSource.addEventListener('click', (e: MouseEvent) => {
          fullPreview({ mode: true, type: 'preview' });
          if ((e.currentTarget as HTMLElement).classList.contains('e-active'))
            defaultRTE.disableToolbarItem(['Bold', 'Italic', 'StrikeThrough',
              '|',
              'Formats', 'OrderedList', 'UnorderedList', '|',

```



```

        'CreateLink', 'Image', 'Undo', 'Redo', 'CreateTable']]);
        (e.currentTarget as
any).parentElement.previousElementSibling.classList.add('e-overlay');
    } else {
        defaultRTE.enableToolbarItem(['Bold', 'Italic', 'StrikeThrough',
'|',
        'Formats', 'OrderedList', 'UnorderedList', '|',
        'CreateLink', 'Image', 'Undo', 'Redo', 'CreateTable']);
        (e.currentTarget as
any).parentElement.previousElementSibling.classList.remove('e-overlay');
    }
});
mdSplit = document.getElementById('MD_Preview') as any;
mdSplit.addEventListener('click', (e: MouseEvent) => {
    if (defaultRTE.element.classList.contains('e-rte-full-screen')) {
        fullPreview({ mode: true, type: '' });
    }
    mdSource.classList.remove('e-active');
    if (!defaultRTE.element.classList.contains('e-rte-full-screen')) {
        defaultRTE.showFullScreen();
    }
});
},
editorMode: 'Markdown',
height: '300px',
toolbarSettings: {
    items: [
        'Bold', 'Italic', 'StrikeThrough', '|', 'Formats', 'OrderedList',
'UnorderedList', '|',
        'CreateLink', 'Image', 'CreateTable', '|',
        {
            template: '<button id="preview-code" class="e-tbar-btn e-control
e-btn e-icon-btn">' +
                '<span class="e-btn-icon e-md-preview e-preview e-
icons"></span></button>',
            tooltipText: 'Preview'
        },
        {
            template: '<button id="MD_Preview" class="e-tbar-btn e-control e-
btn e-icon-btn">' +
                '<span class="e-btn-icon e-view-side e-
icons"></span></button>',
            tooltipText: 'Split Editor'
        },
        'FullScreen', '|', 'Undo', 'Redo'
    ],
    valueTemplate: value
});
defaultRTE.appendTo('#markdownPreview');
});
let mdSource: HTMLElement;
let mdSplit: HTMLElement;
let htmlPreview: HTMLElement;
let defaultRTE: RichTextEditor;
let textArea: HTMLTextAreaElement;
let value: string = `***Overview***

```

The Rich Text Editor component **is WYSIWYG** ("what you see is what you get") editor used to create and edit the content and **return** valid HTML markup or markdown (MD) of the content. The editor provides a standard toolbar to format content **using its** commands. Modular library features to load the necessary functionality on demand. The toolbar contains commands to align the text, insert link, insert image, insert list, undo/redo operation, HTML view, and more.

Key features

- *Mode*: Provides IFRAME and DIV mode.
- *Module*: Modular library to load the necessary functionality on demand.
- *Toolbar*: Provide a fully customizable toolbar.
- *Editing*: HTML view to edit the source directly **for** developers.
- *Third-party Integration*: Supports to integrate third-party library.
- *Preview*: Preview the modified content before saving it.
- *Tools*: Handling images, hyperlinks, video, uploads and more.
- *Undo and Redo*: Undo/redo manager.
- *Lists*: Creates bulleted and numbered list.`;

```
function markDownConversion(): void {
  if (mdSplit.classList.contains('e-active')) {
    const id: string = defaultRTE.getID() + 'html-preview';
    const htmlPreview: any = defaultRTE.element.querySelector('#' + id);
    htmlPreview.innerHTML = marked(((defaultRTE as
any).contentModule.getEditPanel()).value);
  }
}

function fullPreview(e: { [key: string]: string | boolean }): void {
  const id: string = defaultRTE.getID() + 'html-preview';
  htmlPreview = defaultRTE.element.querySelector('#' + id);
  if ((mdSource.classList.contains('e-active') ||
mdSplit.classList.contains('e-active')) && e.mode) {
    mdSource.classList.remove('e-active');
    mdSplit.classList.remove('e-active');
    mdSource.parentElement.title = 'Preview';
    textArea.style.display = 'block';
    textArea.style.width = '100%';
    htmlPreview.style.display = 'none';
  } else {
    mdSource.classList.add('e-active');
    mdSplit.classList.add('e-active');
    if (!htmlPreview) {
      htmlPreview = createElement('div', { className: 'e-content' });
      htmlPreview.id = id;
      (textArea as any).parentNode.appendChild(htmlPreview);
    }
    if (e.type === 'preview') {
      textArea.style.display = 'none'; htmlPreview.classList.add('e-pre-
source');
      htmlPreview.style.width = '100%';
    } else {
      htmlPreview.classList.remove('e-pre-source');
      textArea.style.width = '50%';
      htmlPreview.style.width = '50%';
    }
    htmlPreview.style.display = 'block';
    htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
    mdSource.parentElement.title = 'Code View';
  }
}
```

```

    }
  }
  return (
    <div id="rte-default" className='control-pane'>
      <div className='control-section' id="rtePreview">
        <div className="content-wrapper">
          <div id="markdownPreview"/>
        </div>
      </div>
    </div>
  );
}
export default App;

```

Changing table constants

The Markdown table constants can be changed for the table heading and the column names.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - Markdown to HTML Sample
 */
import { L10n } from '@syncfusion/ej2-base';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { createElement } from '@syncfusion/ej2-base';
import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
L10n.load({
  'en-US': {
    'richtexteditor': {
      'TableColText': 'Cell',
      'TableHeadingText': 'Header'
    }
  }
});
class App extends React.Component {
  mdSource;
  mdSplit;
  htmlPreview;
  defaultRTE;
  textArea;
  value = `***Overview***
  The Rich Text Editor component is WYSIWYG ("what you see is what you get")
  editor used to create and edit the content and return valid HTML markup or
  markdown (MD) of the content. The editor provides a standard toolbar to
  format content using its commands. Modular library features to load the
  necessary functionality on demand. The toolbar contains commands to align the
  text, insert link, insert image, insert list, undo/redo operation, HTML view,
  and more.
  ***Key features***
  - *Mode*: Provides IFRAME and DIV mode.
  - *Module*: Modular library to load the necessary functionality on demand.

```

```

- *Toolbar*: Provide a fully customizable toolbar.
- *Editing*: HTML view to edit the source directly for developers.
- *Third-party Integration*: Supports to integrate third-party library.
- *Preview*: Preview the modified content before saving it.
- *Tools*: Handling images, hyperlinks, video, uploads and more.
- *Undo and Redo*: Undo/redo manager.
- *Lists*: Creates bulleted and numbered list.`;
componentDidMount() {
  this.defaultRTE = new RichTextEditor({
    actionComplete: (e) => {
      if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args))
    {
      this.fullPreview({ mode: true, type: '' });
    }
    else if (!this.mdSplit.parentElement.classList.contains('e-
overlay')) {
      if (e.targetItem === 'Minimize') {
        this.textArea.style.display = 'block';
        this.textArea.style.width = '100%';
        if (this.htmlPreview) {
          this.htmlPreview.style.display = 'none';
        }
        this.mdSplit.classList.remove('e-active');
        this.mdSource.classList.remove('e-active');
      }
      this.markDownConversion();
    }
    setTimeout(() => {
      this.defaultRTE.toolbarModule.refreshToolbarOverflow();
    }, 400);
  },
  created: () => {
    this.textArea = this.defaultRTE.contentModule.getEditPanel();
    this.textArea.addEventListener('keyup', (e) => {
      this.markDownConversion();
    });
    this.mdSource = document.getElementById('preview-code');
    this.mdSource.addEventListener('click', (e) => {
      this.fullPreview({ mode: true, type: 'preview' });
      if (e.currentTarget.classList.contains('e-active')) {
        this.defaultRTE.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
          'Formats', 'OrderedList', 'UnorderedList', '|',
          'CreateLink', 'Image', 'Undo', 'Redo',
'CreateTable']);
e.currentTarget.parentElement.previousElementSibling.classList.add('e-
overlay');
      }
      else {
        this.defaultRTE.enableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
          'Formats', 'OrderedList', 'UnorderedList', '|',
          'CreateLink', 'Image', 'Undo', 'Redo',
'CreateTable']);

```

```

e.currentTarget.parentElement.previousElementSibling.classList.remove('e-
overlay');
    }
    });
    this.mdSplit = document.getElementById('MD_Preview');
    this.mdSplit.addEventListener('click', (e) => {
        if (this.defaultRTE.element.classList.contains('e-rte-
full-screen')) {
            this.fullPreview({ mode: true, type: '' });
        }
        this.mdSource.classList.remove('e-active');
        if (!this.defaultRTE.element.classList.contains('e-rte-
full-screen')) {
            this.defaultRTE.showFullScreen();
        }
    });
},
editorMode: 'Markdown',
height: '300px',
toolbarSettings: {
    items: [
        'Bold', 'Italic', 'StrikeThrough', '|', 'Formats',
'OrderedList', 'UnorderedList', '|',
        'CreateLink', 'Image', 'CreateTable', '|',
        {
            template: '<button id="preview-code" class="e-tbar-
btn e-control e-btn e-icon-btn">' +
                '<span class="e-btn-icon e-md-preview e-preview
e-icons"></span></button>',
            tooltipText: 'Preview'
        },
        {
            template: '<button id="MD_Preview" class="e-tbar-btn
e-control e-btn e-icon-btn">' +
                '<span class="e-btn-icon e-view-side e-
icons"></span></button>',
            tooltipText: 'Split Editor'
        },
        'FullScreen', '|', 'Undo', 'Redo'
    ]
},
valueTemplate: this.value,
});
this.defaultRTE.appendTo('#markdownPreview');
}
markDownConversion() {
    if (this.mdSplit.classList.contains('e-active')) {
        const id = this.defaultRTE.getID() + 'html-preview';
        const htmlPreview = this.defaultRTE.element.querySelector('#' +
id);
        htmlPreview.innerHTML =
marked((this.defaultRTE.contentModule.getEditPanel()).value);
    }
}
fullPreview(e) {
    const id = this.defaultRTE.getID() + 'html-preview';

```

```

        this.htmlPreview = this.defaultRTE.element.querySelector('#' + id);
        if ((this.mdSource.classList.contains('e-active') ||
this.mdSplit.classList.contains('e-active')) && e.mode) {
            this.mdSource.classList.remove('e-active');
            this.mdSplit.classList.remove('e-active');
            this.mdSource.parentElement.title = 'Preview';
            this.textArea.style.display = 'block';
            this.textArea.style.width = '100%';
            this.htmlPreview.style.display = 'none';
        }
        else {
            this.mdSource.classList.add('e-active');
            this.mdSplit.classList.add('e-active');
            if (!this.htmlPreview) {
                this.htmlPreview = createElement('div', { className: 'e-
content' });
                this.htmlPreview.id = id;
                this.textArea.parentNode.appendChild(this.htmlPreview);
            }
            if (e.type === 'preview') {
                this.textArea.style.display = 'none';
                this.htmlPreview.classList.add('e-pre-source');
                this.htmlPreview.style.width = '100%';
            }
            else {
                this.htmlPreview.classList.remove('e-pre-source');
                this.textArea.style.width = '50%';
                this.htmlPreview.style.width = '50%';
            }
            this.htmlPreview.style.display = 'block';
            this.htmlPreview.innerHTML =
marked((this.defaultRTE.contentModule.getEditPanel()).value);
            this.mdSource.parentElement.title = 'Code View';
        }
    }
    render() {
        return (<div id="rte-default" className='control-pane'>
        <div className='control-section' id="rtePreview">
            <div className="content-wrapper">
                <div id="markdownPreview"/>
            </div>
        </div>
        </div>);
    }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Markdown to HTML Sample
 */
import { L10n } from '@syncfusion/ej2-base';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { createElement, KeyboardEventArgs } from '@syncfusion/ej2-base';

```

```

import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from
'@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
L10n.load({
  'en-US': {
    'richtexteditor': {
      'TableColText': 'Cell',
      'TableHeadingText': 'Header'
    }
  }
});
class App extends React.Component<{}, {}> {
  public mdSource: HTMLElement;
  public mdSplit: HTMLElement;
  public htmlPreview: HTMLElement;
  public defaultRTE: RichTextEditor;
  public textArea: HTMLTextAreaElement;
  public value: string = `***Overview***
  The Rich Text Editor component is WYSIWYG ("what you see is what you get")
  editor used to create and edit the content and return valid HTML markup or
  markdown (MD) of the content. The editor provides a standard toolbar to
  format content using its commands. Modular library features to load the
  necessary functionality on demand. The toolbar contains commands to align the
  text, insert link, insert image, insert list, undo/redo operation, HTML view,
  and more.
  ***Key features***
  - *Mode*: Provides IFRAME and DIV mode.
  - *Module*: Modular library to load the necessary functionality on demand.
  - *Toolbar*: Provide a fully customizable toolbar.
  - *Editing*: HTML view to edit the source directly for developers.
  - *Third-party Integration*: Supports to integrate third-party library.
  - *Preview*: Preview the modified content before saving it.
  - *Tools*: Handling images, hyperlinks, video, uploads and more.
  - *Undo and Redo*: Undo/redo manager.
  - *Lists*: Creates bulleted and numbered list.`;
  public componentDidMount(): void {
    this.defaultRTE = new RichTextEditor({
      actionComplete: (e: any) => {
        if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args)) {
          this.fullPreview({ mode: true, type: ' ' });
        } else if (!(this.mdSplit as
any).parentElement.classList.contains('e-overlay')) {
          if (e.targetItem === 'Minimize') {
            this.textArea.style.display = 'block';
            this.textArea.style.width = '100%';
            if (this.htmlPreview) { this.htmlPreview.style.display = 'none';
          }

          this.mdSplit.classList.remove('e-active');
          this.mdSource.classList.remove('e-active');
        }
        this.markDownConversion();
      }
    });
    setTimeout(() => {
      this.defaultRTE.toolbarModule.refreshToolbarOverflow();
    }, 400);
  },
},

```

```

        created: () => {
            this.textArea = (this.defaultRTE as
any).contentModule.getEditPanel();
            this.textArea.addEventListener('keyup', (e: KeyboardEventArgs) => {
                this.markDownConversion();
            });
            this.mdSource = document.getElementById('preview-code') as any;
            this.mdSource.addEventListener('click', (e: MouseEvent) => {
                this.fullPreview({ mode: true, type: 'preview' });
                if ((e.currentTarget as HTMLElement).classList.contains('e-active'))
            {
                this.defaultRTE.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
                'Formats', 'OrderedList', 'UnorderedList', '|',
                'CreateLink', 'Image', 'Undo', 'Redo', 'CreateTable']);
                (e.currentTarget as
any).parentElement.previousElementSibling.classList.add('e-overlay');
            } else {
                this.defaultRTE.enableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
                'Formats', 'OrderedList', 'UnorderedList', '|',
                'CreateLink', 'Image', 'Undo', 'Redo', 'CreateTable']);
                (e.currentTarget as
any).parentElement.previousElementSibling.classList.remove('e-overlay');
            }
        });
            this.mdSplit = document.getElementById('MD_Preview') as any;
            this.mdSplit.addEventListener('click', (e: MouseEvent) => {
                if (this.defaultRTE.element.classList.contains('e-rte-full-screen'))
            {
                this.fullPreview({ mode: true, type: '' });
            }
                this.mdSource.classList.remove('e-active');
                if (!this.defaultRTE.element.classList.contains('e-rte-full-screen'))
            {
                this.defaultRTE.showFullScreen();
            }
            });
        },
        editorMode: 'Markdown',
        height: '300px',
        toolbarSettings: {
            items: [
                'Bold', 'Italic', 'StrikeThrough', '|', 'Formats', 'OrderedList',
'UnorderedList', '|',
                'CreateLink', 'Image', 'CreateTable', '|',
                {
                    template: '<button id="preview-code" class="e-tbar-btn e-control
e-btn e-icon-btn">' +
                        '<span class="e-btn-icon e-md-preview e-preview e-
icons"></span></button>',
                    tooltipText: 'Preview'
                },
                {
                    template: '<button id="MD_Preview" class="e-tbar-btn e-control e-
btn e-icon-btn">' +

```



```

        '<span class="e-btn-icon e-view-side e-
icons"></span></button>',
        tooltipText: 'Split Editor'
    },
    'FullScreen', '|', 'Undo', 'Redo']
    },
    valueTemplate: this.value,
    });
    this.defaultRTE.appendTo('#markdownPreview');
}
public markDownConversion(): void {
    if (this.mdSplit.classList.contains('e-active')) {
        const id: string = this.defaultRTE.getID() + 'html-preview';
        const htmlPreview: HTMLElement =
this.defaultRTE.element.querySelector('#' + id) as any;
        htmlPreview.innerHTML = marked(((this.defaultRTE as
any).contentModule.getEditPanel()).value);
    }
}
public fullPreview(e: { [key: string]: string | boolean }): void {
    const id: string = this.defaultRTE.getID() + 'html-preview';
    this.htmlPreview = this.defaultRTE.element.querySelector('#' + id);
    if ((this.mdSource.classList.contains('e-active') ||
this.mdSplit.classList.contains('e-active')) && e.mode) {
        this.mdSource.classList.remove('e-active');
        this.mdSplit.classList.remove('e-active');
        this.mdSource.parentElement.title = 'Preview';
        this.textArea.style.display = 'block';
        this.textArea.style.width = '100%';
        this.htmlPreview.style.display = 'none';
    } else {
        this.mdSource.classList.add('e-active');
        this.mdSplit.classList.add('e-active');
        if (!this.htmlPreview) {
            this.htmlPreview = createElement('div', { className: 'e-content' });
            this.htmlPreview.id = id;
            (this.textArea as any).parentNode.appendChild(this.htmlPreview);
        }
        if (e.type === 'preview') {
            this.textArea.style.display = 'none';
            this.htmlPreview.classList.add('e-pre-source');
            this.htmlPreview.style.width = '100%';
        } else {
            this.htmlPreview.classList.remove('e-pre-source');
            this.textArea.style.width = '50%';
            this.htmlPreview.style.width = '50%';
        }
        this.htmlPreview.style.display = 'block';
        this.htmlPreview.innerHTML = marked(((this as
any).defaultRTE.contentModule.getEditPanel()).value);
        this.mdSource.parentElement.title = 'Code View';
    }
}
public render() {
    return (
        <div id="rte-default" className='control-pane'>
            <div className='control-section' id="rtePreview">

```

```

        <div className="content-wrapper">
            <div id="markdownPreview"/>
        </div>
    </div>
</div>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Markdown to HTML Sample
 */
import { L10n } from '@syncfusion/ej2-base';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { createElement } from '@syncfusion/ej2-base';
import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
import { useEffect } from 'react';
L10n.load({
    'en-US': {
        'richtexteditor': {
            'TableColText': 'Cell',
            'TableHeadingText': 'Header'
        }
    }
});
function App() {
    useEffect(() => {
        defaultRTE = new RichTextEditor({
            actionComplete: (e) => {
                if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args))
                {
                    fullPreview({ mode: true, type: '' });
                }
                else if (!mdSplit.parentElement.classList.contains('e-overlay')) {
                    if (e.targetItem === 'Minimize') {
                        textArea.style.display = 'block';
                        textArea.style.width = '100%';
                        if (htmlPreview) {
                            htmlPreview.style.display = 'none';
                        }
                        mdSplit.classList.remove('e-active');
                        mdSource.classList.remove('e-active');
                    }
                    markDownConversion();
                }
                setTimeout(() => {
                    defaultRTE.toolbarModule.refreshToolbarOverflow();

```

```

        }, 400);
    },
    created: () => {
        textArea = defaultRTE.contentModule.getEditPanel();
        textArea.addEventListener('keyup', (e) => {
            markDownConversion();
        });
        mdSource = document.getElementById('preview-code');
        mdSource.addEventListener('click', (e) => {
            fullPreview({ mode: true, type: 'preview' });
            if (e.currentTarget.classList.contains('e-active')) {
                defaultRTE.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
                                'Formats', 'OrderedList', 'UnorderedList', '|',
                                'CreateLink', 'Image', 'Undo', 'Redo',
'CreateTable']);
            }
            e.currentTarget.parentElement.previousElementSibling.classList.add('e-
overlay');
        }
        else {
            defaultRTE.enableToolbarItem(['Bold', 'Italic',
'StrikeThrough', '|',
                                'Formats', 'OrderedList', 'UnorderedList', '|',
                                'CreateLink', 'Image', 'Undo', 'Redo',
'CreateTable']);
        }
        e.currentTarget.parentElement.previousElementSibling.classList.remove('e-
overlay');
    }
    });
    mdSplit = document.getElementById('MD_Preview');
    mdSplit.addEventListener('click', (e) => {
        if (defaultRTE.element.classList.contains('e-rte-full-
screen')) {
            fullPreview({ mode: true, type: '' });
        }
        mdSource.classList.remove('e-active');
        if (!defaultRTE.element.classList.contains('e-rte-full-
screen')) {
            defaultRTE.showFullScreen();
        }
    });
    },
    editorMode: 'Markdown',
    height: '300px',
    toolbarSettings: {
        items: [
            'Bold', 'Italic', 'StrikeThrough', '|', 'Formats',
'OrderedList', 'UnorderedList', '|',
            'CreateLink', 'Image', 'CreateTable', '|',
            {
                template: '<button id="preview-code" class="e-tbar-
btn e-control e-btn e-icon-btn">' +
                    '<span class="e-btn-icon e-md-preview e-preview
e-icons"></span></button>',
                tooltipText: 'Preview'
            }
        ]
    }

```

```

        },
        {
            template: '<button id="MD_Preview" class="e-tbar-btn
e-control e-btn e-icon-btn">' +
                '<span class="e-btn-icon e-view-side e-
icons"></span></button>',
            tooltipText: 'Split Editor'
        },
        'FullScreen', '|', 'Undo', 'Redo'
    ]
},
valueTemplate: value,
));
defaultRTE.appendTo('#markdownPreview');
});
let mdSource;
let mdSplit;
let htmlPreview;
let defaultRTE;
let textArea;
let value = `***Overview***
The Rich Text Editor component is WYSIWYG ("what you see is what you get")
editor used to create and edit the content and return valid HTML markup or
markdown (MD) of the content. The editor provides a standard toolbar to
format content using its commands. Modular library features to load the
necessary functionality on demand. The toolbar contains commands to align the
text, insert link, insert image, insert list, undo/redo operation, HTML view,
and more.
***Key features***
- *Mode*: Provides IFRAME and DIV mode.
- *Module*: Modular library to load the necessary functionality on demand.
- *Toolbar*: Provide a fully customizable toolbar.
- *Editing*: HTML view to edit the source directly for developers.
- *Third-party Integration*: Supports to integrate third-party library.
- *Preview*: Preview the modified content before saving it.
- *Tools*: Handling images, hyperlinks, video, uploads and more.
- *Undo and Redo*: Undo/redo manager.
- *Lists*: Creates bulleted and numbered list.`;
function markDownConversion() {
    if (mdSplit.classList.contains('e-active')) {
        const id = defaultRTE.getID() + 'html-preview';
        const htmlPreview = defaultRTE.element.querySelector('#' + id);
        htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
    }
}
function fullPreview(e) {
    const id = defaultRTE.getID() + 'html-preview';
    htmlPreview = defaultRTE.element.querySelector('#' + id);
    if ((mdSource.classList.contains('e-active') ||
mdSplit.classList.contains('e-active')) && e.mode) {
        mdSource.classList.remove('e-active');
        mdSplit.classList.remove('e-active');
        mdSource.parentElement.title = 'Preview';
        textArea.style.display = 'block';
        textArea.style.width = '100%';
        htmlPreview.style.display = 'none';
    }
}

```

```

    }
    else {
      mdSource.classList.add('e-active');
      mdSplit.classList.add('e-active');
      if (!htmlPreview) {
        htmlPreview = createElement('div', { className: 'e-content'
      });

      htmlPreview.id = id;
      textArea.parentNode.appendChild(htmlPreview);
    }
    if (e.type === 'preview') {
      textArea.style.display = 'none';
      htmlPreview.classList.add('e-pre-source');
      htmlPreview.style.width = '100%';
    }
    else {
      htmlPreview.classList.remove('e-pre-source');
      textArea.style.width = '50%';
      htmlPreview.style.width = '50%';
    }
    htmlPreview.style.display = 'block';
    htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
    mdSource.parentElement.title = 'Code View';
  }
}
return (<div id="rte-default" className='control-pane'>
  <div className='control-section' id="rtePreview">
    <div className="content-wrapper">
      <div id="markdownPreview"/>
    </div>
  </div>
</div>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Markdown to HTML Sample
 */
import { L10n } from '@syncfusion/ej2-base';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { createElement, KeyboardEventArgs } from '@syncfusion/ej2-base';
import { Image, Link, MarkdownEditor, RichTextEditor, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
RichTextEditor.Inject(Image, Link, MarkdownEditor, Toolbar);
import * as React from 'react';
import { useEffect } from "react";
L10n.load({
  'en-US': {
    'richtexteditor': {
      'TableColText': 'Cell',
      'TableHeadingText': 'Header'
    }
  }
})

```

```

});
function App() {
  useEffect(() => {
    defaultRTE = new RichTextEditor({
      actionComplete: (e: any) => {
        if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args)) {
          fullPreview({ mode: true, type: '' });
        } else if (!(mdSplit as any).parentElement.classList.contains('e-
overlay')) {
          if (e.targetItem === 'Minimize') {
            textArea.style.display = 'block';
            textArea.style.width = '100%';
            if (htmlPreview) { htmlPreview.style.display = 'none'; }
            mdSplit.classList.remove('e-active');
            mdSource.classList.remove('e-active');
          }
          markDownConversion();
        }
        setTimeout(() => {
          defaultRTE.toolbarModule.refreshToolbarOverflow();
        }, 400);
      },
      created: () => {
        textArea = (defaultRTE as any).contentModule.getEditPanel();
        textArea.addEventListener('keyup', (e: KeyboardEventArgs) => {
          markDownConversion();
        });
        mdSource = document.getElementById('preview-code') as any;
        mdSource.addEventListener('click', (e: MouseEvent) => {
          fullPreview({ mode: true, type: 'preview' });
          if ((e.currentTarget as HTMLElement).classList.contains('e-active'))
          {
            defaultRTE.disableToolbarItem(['Bold', 'Italic', 'StrikeThrough',
            '|',
            'Formats', 'OrderedList', 'UnorderedList', '|',
            'CreateLink', 'Image', 'Undo', 'Redo', 'CreateTable']);
            (e.currentTarget as
any).parentElement.previousElementSibling.classList.add('e-overlay');
          } else {
            defaultRTE.enableToolbarItem(['Bold', 'Italic', 'StrikeThrough',
            '|',
            'Formats', 'OrderedList', 'UnorderedList', '|',
            'CreateLink', 'Image', 'Undo', 'Redo', 'CreateTable']);
            (e.currentTarget as
any).parentElement.previousElementSibling.classList.remove('e-overlay');
          }
        });
        mdSplit = document.getElementById('MD_Preview') as any;
        mdSplit.addEventListener('click', (e: MouseEvent) => {
          if (defaultRTE.element.classList.contains('e-rte-full-screen')) {
            fullPreview({ mode: true, type: '' });
          }
          mdSource.classList.remove('e-active');
          if (!defaultRTE.element.classList.contains('e-rte-full-screen')) {
            defaultRTE.showFullScreen();
          }
        });
      }
    });
  });
}

```

```

    },
    editorMode: 'Markdown',
    height: '300px',
    toolbarSettings: {
      items: [
        'Bold', 'Italic', 'StrikeThrough', '|', 'Formats', 'OrderedList',
        'UnorderedList', '|',
        'CreateLink', 'Image', 'CreateTable', '|',
        {
          template: '<button id="preview-code" class="e-tbar-btn e-control e-btn e-icon-btn">' +
            '<span class="e-btn-icon e-md-preview e-preview e-icons"></span></button>',
          tooltipText: 'Preview'
        },
        {
          template: '<button id="MD_Preview" class="e-tbar-btn e-control e-btn e-icon-btn">' +
            '<span class="e-btn-icon e-view-side e-icons"></span></button>',
          tooltipText: 'Split Editor'
        },
        'FullScreen', '|', 'Undo', 'Redo'
      ],
      valueTemplate: value,
    });
    defaultRTE.appendTo('#markdownPreview');
  });
  let mdSource: HTMLElement;
  let mdSplit: HTMLElement;
  let htmlPreview: HTMLElement;
  let defaultRTE: RichTextEditor;
  let textArea: HTMLTextAreaElement;
  let value: string = `***Overview***`
  The Rich Text Editor component is WYSIWYG ("what you see is what you get")
  editor used to create and edit the content and return valid HTML markup or
  markdown (MD) of the content. The editor provides a standard toolbar to
  format content using its commands. Modular library features to load the
  necessary functionality on demand. The toolbar contains commands to align the
  text, insert link, insert image, insert list, undo/redo operation, HTML view,
  and more.
  ***Key features***
  - *Mode*: Provides IFRAME and DIV mode.
  - *Module*: Modular library to load the necessary functionality on demand.
  - *Toolbar*: Provide a fully customizable toolbar.
  - *Editing*: HTML view to edit the source directly for developers.
  - *Third-party Integration*: Supports to integrate third-party library.
  - *Preview*: Preview the modified content before saving it.
  - *Tools*: Handling images, hyperlinks, video, uploads and more.
  - *Undo and Redo*: Undo/redo manager.
  - *Lists*: Creates bulleted and numbered list.`;
  function markDownConversion(): void {
    if (mdSplit.classList.contains('e-active')) {
      const id: string = defaultRTE.getID() + 'html-preview';
      const htmlPreview: HTMLElement = defaultRTE.element.querySelector('#' +
id) as any;

```

```

        htmlPreview.innerHTML = marked(((defaultRTE as
any).contentModule.getEditPanel()).value);
    }
}
function fullPreview(e: { [key: string]: string | boolean }): void {
    const id: string = defaultRTE.getID() + 'html-preview';
    htmlPreview = defaultRTE.element.querySelector('#' + id);
    if ((mdSource.classList.contains('e-active') ||
mdSplit.classList.contains('e-active')) && e.mode) {
        mdSource.classList.remove('e-active');
        mdSplit.classList.remove('e-active');
        mdSource.parentElement.title = 'Preview';
        textArea.style.display = 'block';
        textArea.style.width = '100%';
        htmlPreview.style.display = 'none';
    } else {
        mdSource.classList.add('e-active');
        mdSplit.classList.add('e-active');
        if (!htmlPreview) {
            htmlPreview = createElement('div', { className: 'e-content' });
            htmlPreview.id = id;
            (textArea as any).parentNode.appendChild(htmlPreview);
        }
        if (e.type === 'preview') {
            textArea.style.display = 'none'; htmlPreview.classList.add('e-pre-
source');
            htmlPreview.style.width = '100%';
        } else {
            htmlPreview.classList.remove('e-pre-source');
            textArea.style.width = '50%';
            htmlPreview.style.width = '50%';
        }
        htmlPreview.style.display = 'block';
        htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
        mdSource.parentElement.title = 'Code View';
    }
}
return (
    <div id="rte-default" className='control-pane'>
        <div className='control-section' id="rtePreview">
            <div className="content-wrapper">
                <div id="markdownPreview"/>
            </div>
        </div>
    </div>
);
}
export default App;

```

Custom format

The Rich Text Editor allows you to customize the markdown syntax by overriding its default syntax. Configure the customized markdown syntax using the [formatter](#) property.

This sample demonstrates how to customize tags of markdown formatting.

For example, apply **+** to Unordered list, apply **1., 2., 3.** to Ordered list, for bold, , and for italic .

[Class-component]

APP.JSX

```
{% raw %}
/**
 * Rich Text Editor - Markdown - Custom Format Sample
 */
import { createElement } from '@syncfusion/ej2-base';
import { Image, Inject, Link, MarkdownEditor, MarkdownFormatter,
QuickToolbar, RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  mdSource;
  mdSplit;
  htmlPreview;
  defaultRTE;
  textArea;
  // set the value to Rich Text Editor
  template = `The sample is configured with customized markdown syntax
using the __formatter__ property. Type the content and click the toolbar item
to view customized markdown syntax. For unordered list, you need to add a
plus sign before the word (e.g., + list1). Or To make a phrase bold, you need
to add two underscores before and after the phrase (e.g., __this text is
bold__).`;
  // Rich Text Editor items list
  items = ['Bold', 'Italic', 'StrikeThrough', '|',
    'Formats', 'OrderedList', 'UnorderedList', '|',
    'CreateLink', 'Image', '|',
    {
      template: '<button id="preview-code" class="e-tbar-btn e-control
e-btn e-icon-btn">' +
        '<span class="e-btn-icon e-icons e-md-
preview"></span></button>',
      tooltipText: 'Preview',
    }, 'Undo', 'Redo'];
  mdsources;
  mdPreview;
  // Rich Text Editor ToolbarSettings
  toolbarSettings = {
    items: this.items
  };
  formatter = new MarkdownFormatter({
    formatTags: {
      'Blockquote': '> '
    },
    listTags: { 'OL': '1., 2., 3.', 'UL': '+ ' },
    selectionTags: { 'Bold': '__', 'Italic': '_' }
  });
  markDownConversion() {
    if (this.mdsources.classList.contains('e-active')) {
      const id = this.rteObj.getID() + 'html-view';
      const htmlPreview = this.rteObj.element.querySelector('#' + id);
    }
  }
}
```

```

        htmlPreview.innerHTML =
marked((this.rteObj.contentModule.getEditPanel()).value);
    }
}
fullPreview(e) {
    const id = this.defaultRTE.getID() + 'html-preview';
    this.htmlPreview = this.defaultRTE.element.querySelector('#' + id);
    if ((this.mdSource.classList.contains('e-active') ||
this.mdSplit.classList.contains('e-active')) && e.mode) {
        this.mdSource.classList.remove('e-active');
        this.mdSplit.classList.remove('e-active');
        this.mdSource.parentElement.title = 'Preview';
        this.textArea.style.display = 'block';
        this.textArea.style.width = '100%';
        this.htmlPreview.style.display = 'none';
    }
    else {
        this.mdSource.classList.add('e-active');
        this.mdSplit.classList.add('e-active');
        if (!this.htmlPreview) {
            this.htmlPreview = createElement('div', { className: 'e-
content' });
            this.htmlPreview.id = id;
            this.textArea.parentNode.appendChild(this.htmlPreview);
        }
        if (e.type === 'preview') {
            this.textArea.style.display = 'none';
            this.htmlPreview.classList.add('e-pre-source');
            this.htmlPreview.style.width = '100%';
        }
        else {
            this.htmlPreview.classList.remove('e-pre-source');
            this.textArea.style.width = '50%';
            this.htmlPreview.style.width = '50%';
        }
        this.htmlPreview.style.display = 'block';
        this.htmlPreview.innerHTML =
marked((this.defaultRTE.contentModule.getEditPanel()).value);
        this.mdSource.parentElement.title = 'Code View';
    }
}
render() {
    return (<RichTextEditorComponent id="markdownRTE"
ref={ (richtexteditor) => { this.rteObj = richtexteditor; }} height='260px'
editorMode='Markdown' formatter={this.formatter}
valueTemplate={this.template} toolbarSettings={this.toolbarSettings}>
    <Inject services={[MarkdownEditor, Toolbar, Image, Link,
QuickToolbar]}/>
    </RichTextEditorComponent>);
}
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Markdown - Custom Format Sample
 */
import { createElement, KeyboardEventArgs } from '@syncfusion/ej2-base';
import { Image, Inject, Link, MarkdownEditor, MarkdownFormatter,
QuickToolbar, RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    public rteObj: RichTextEditorComponent;
    public mdSource: HTMLInputElement;
    public mdSplit: HTMLInputElement;
    public htmlPreview: HTMLInputElement;
    public defaultRTE: RichTextEditor;
    public textArea: HTMLTextAreaElement;
    // set the value to Rich Text Editor
    public template: string = `The sample is configured with customized
markdown syntax using the __formatter__ property. Type the content and click
the toolbar item to view customized markdown syntax. For unordered list, you
need to add a plus sign before the word (e.g., + list1). Or To make a phrase
bold, you need to add two underscores before and after the phrase (e.g.,
__this text is bold__).`;
    // Rich Text Editor items list
    public items: object = ['Bold', 'Italic', 'StrikeThrough', '|',
'Formats', 'OrderedList', 'UnorderedList', '|',
'CreateLink', 'Image', '|',
{
    template: '<button id="preview-code" class="e-tbar-btn e-control e-
btn e-icon-btn">' +
        '<span class="e-btn-icon e-icons e-md-preview"></span></button>',
    tooltipText: 'Preview',
}, 'Undo', 'Redo'];
    public mdsSource: HTMLInputElement;
    public mdPreview: HTMLInputElement;
    // Rich Text Editor ToolbarSettings
    public toolbarSettings: object = {
        items: this.items
    };
    public formatter = new MarkdownFormatter({
        formatTags: {
            'Blockquote': '> '
        },
        listTags: { 'OL': '1., 2., 3.', 'UL': '+ ' },
        selectionTags: { 'Bold': '__', 'Italic': '_' }
    });
    public markDownConversion(): void {
        if (this.mdsSource.classList.contains('e-active')) {
            const id: string = (this.rteObj as any).getID() + 'html-view';
            const htmlPreview: HTMLInputElement = (this.rteObj as
any).element.querySelector('#' + id);
            htmlPreview.innerHTML = marked(((this.rteObj as
any).contentModule.getEditPanel()).value);
        }
    }
    public fullPreview(e: { [key: string]: string | boolean }): void {
        const id: string = this.defaultRTE.getID() + 'html-preview';

```

```

        this.htmlPreview = this.defaultRTE.element.querySelector('#' + id);
        if ((this.mdSource.classList.contains('e-active') ||
this.mdSplit.classList.contains('e-active')) && e.mode) {
            this.mdSource.classList.remove('e-active');
            this.mdSplit.classList.remove('e-active');
            this.mdSource.parentElement.title = 'Preview';
            this.textArea.style.display = 'block';
            this.textArea.style.width = '100%';
            this.htmlPreview.style.display = 'none';
        } else {
            this.mdSource.classList.add('e-active');
            this.mdSplit.classList.add('e-active');
            if (!this.htmlPreview) {
                this.htmlPreview = createElement('div', { className: 'e-content'
});
                this.htmlPreview.id = id;
                (this.textArea as any).parentNode.appendChild(this.htmlPreview);
            }
            if (e.type === 'preview') {
                this.textArea.style.display = 'none';
                this.htmlPreview.classList.add('e-pre-source');
                this.htmlPreview.style.width = '100%';
            } else {
                this.htmlPreview.classList.remove('e-pre-source');
                this.textArea.style.width = '50%';
                this.htmlPreview.style.width = '50%';
            }
            this.htmlPreview.style.display = 'block';
            this.htmlPreview.innerHTML = marked(((this as
any).defaultRTE.contentModule.getEditPanel()).value);
            this.mdSource.parentElement.title = 'Code View';
        }
    }
    public render() {
        return (
            <RichTextEditorComponent id="markdownRTE"
                ref={(richtexteditor) => { this.rteObj = richtexteditor! }}
                height='260px' editorMode='Markdown'
                formatter= {this.formatter}
                valueTemplate={this.template}
                toolbarSettings={this.toolbarSettings} >
                <Inject services={[MarkdownEditor, Toolbar, Image, Link,
QuickToolbar]} />
            </RichTextEditorComponent>
        );
    }
}
export default App;
{% endraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
/**

```

```

* Rich Text Editor - Markdown - Custom Format Sample
*/
import { createElement } from '@syncfusion/ej2-base';
import { Image, Inject, Link, MarkdownEditor, MarkdownFormatter,
QuickToolbar, RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    let rteObj;
    let mdSource;
    let mdSplit;
    let htmlPreview;
    let defaultRTE;
    let textArea;
    // set the value to Rich Text Editor
    let template = `The sample is configured with customized markdown syntax
using the __formatter__ property. Type the content and click the toolbar item
to view customized markdown syntax. For unordered list, you need to add a
plus sign before the word (e.g., + list1). Or To make a phrase bold, you need
to add two underscores before and after the phrase (e.g., __this text is
bold __).`;
    // Rich Text Editor items list
    let items = ['Bold', 'Italic', 'StrikeThrough', '|',
        'Formats', 'OrderedList', 'UnorderedList', '|',
        'CreateLink', 'Image', '|',
        {
            template: '<button id="preview-code" class="e-tbar-btn e-control
e-btn e-icon-btn">' +
                '<span class="e-btn-icon e-icons e-md-
preview"></span></button>',
            tooltipText: 'Preview',
        }, 'Undo', 'Redo'];
    let mdsources;
    let mdPreview;
    // Rich Text Editor ToolbarSettings
    let toolbarSettings = {
        items: items
    };
    let formatter = new MarkdownFormatter({
        formatTags: {
            'Blockquote': '>'
        },
        listTags: { 'OL': '1., 2., 3.', 'UL': '+ ' },
        selectionTags: { 'Bold': '__', 'Italic': '_' }
    });
    function markDownConversion() {
        if (mdsource.classList.contains('e-active')) {
            const id = rteObj.getID() + 'html-view';
            const htmlPreview = rteObj.element.querySelector('#' + id);
            htmlPreview.innerHTML =
marked((rteObj.contentModule.getEditPanel()).value);
        }
    }
    function fullPreview(e) {
        const id = defaultRTE.getID() + 'html-preview';
        htmlPreview = defaultRTE.element.querySelector('#' + id);
    }
}

```

```

        if ((mdSource.classList.contains('e-active') ||
mdSplit.classList.contains('e-active')) && e.mode) {
            mdSource.classList.remove('e-active');
            mdSplit.classList.remove('e-active');
            mdSource.parentElement.title = 'Preview';
            textArea.style.display = 'block';
            textArea.style.width = '100%';
            htmlPreview.style.display = 'none';
        }
        else {
            mdSource.classList.add('e-active');
            mdSplit.classList.add('e-active');
            if (!htmlPreview) {
                htmlPreview = createElement('div', { className: 'e-content '
});

                htmlPreview.id = id;
                textArea.parentNode.appendChild(htmlPreview);
            }
            if (e.type === 'preview') {
                textArea.style.display = 'none';
                htmlPreview.classList.add('e-pre-source');
                htmlPreview.style.width = '100%';
            }
            else {
                htmlPreview.classList.remove('e-pre-source');
                textArea.style.width = '50%';
                htmlPreview.style.width = '50%';
            }
            htmlPreview.style.display = 'block';
            htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
            mdSource.parentElement.title = 'Code View';
        }
    }

    return (<RichTextEditorComponent id="markdownRTE" ref={ (richtexteditor)
=> { rteObj = richtexteditor; }} height='260px' editorMode='Markdown'
formatter={formatter} valueTemplate={template}
toolbarSettings={toolbarSettings}>
        <Inject services={[MarkdownEditor, Toolbar, Image, Link,
QuickToolbar]}/>
    </RichTextEditorComponent>;
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Markdown - Custom Format Sample
 */
import { createElement, KeyboardEventArgs } from '@syncfusion/ej2-base';
import { Image, Inject, Link, MarkdownEditor, MarkdownFormatter,
QuickToolbar, RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';

```

```

function App() {
  let rteObj: RichTextEditorComponent;
  let mdSource: HTMLElement;
  let mdSplit: HTMLElement;
  let htmlPreview: HTMLElement;
  let defaultRTE: RichTextEditor;
  let textArea: HTMLTextAreaElement;
  // set the value to Rich Text Editor
  let template: string = `The sample is configured with customized
markdown syntax using the __formatter__ property. Type the content and click
the toolbar item to view customized markdown syntax. For unordered list, you
need to add a plus sign before the word (e.g., + list1). Or To make a phrase
bold, you need to add two underscores before and after the phrase (e.g.,
__this text is bold__).`;
  // Rich Text Editor items list
  let items: object = ['Bold', 'Italic', 'StrikeThrough', '|',
    'Formats', 'OrderedList', 'UnorderedList', '|',
    'CreateLink', 'Image', '|',
    {
      template: '<button id="preview-code" class="e-tbar-btn e-control e-
btn e-icon-btn">' +
        '<span class="e-btn-icon e-icons e-md-
preview"></span></button>',
      tooltipText: 'Preview',
    }, 'Undo', 'Redo'];
  let mdsources: HTMLElement;
  let mdPreview: HTMLElement;
  // Rich Text Editor ToolbarSettings
  let toolbarSettings: object = {
    items: items
  };
  let formatter = new MarkdownFormatter({
    formatTags: {
      'Blockquote': '> '
    },
    listTags: { 'OL': '1., 2., 3.', 'UL': '+ ' },
    selectionTags: { 'Bold': '__', 'Italic': '_' }
  });
  function markDownConversion(): void {
    if (mdsource.classList.contains('e-active')) {
      const id: string = (rteObj as any).getID() + 'html-view';
      const htmlPreview: HTMLElement = (rteObj as
any).element.querySelector('#' + id);
      htmlPreview.innerHTML = marked(((rteObj as
any).contentModule.getEditPanel()).value);
    }
  }
  function fullPreview(e: { [key: string]: string | boolean }): void {
    const id: string = defaultRTE.getID() + 'html-preview';
    htmlPreview = defaultRTE.element.querySelector('#' + id);
    if ((mdSource.classList.contains('e-active') ||
mdSplit.classList.contains('e-active')) && e.mode) {
      mdSource.classList.remove('e-active');
      mdSplit.classList.remove('e-active');
      mdSource.parentElement.title = 'Preview';
      textArea.style.display = 'block';
      textArea.style.width = '100%';
    }
  }
}

```

```

        htmlPreview.style.display = 'none';
    } else {
        mdSource.classList.add('e-active');
        mdSplit.classList.add('e-active');
        if (!htmlPreview) {
            htmlPreview = createElement('div', { className: 'e-content' });
            htmlPreview.id = id;
            (textArea as any).parentNode.appendChild(htmlPreview);
        }
        if (e.type === 'preview') {
            textArea.style.display = 'none'; htmlPreview.classList.add('e-pre-
source');
            htmlPreview.style.width = '100%';
        } else {
            htmlPreview.classList.remove('e-pre-source');
            textArea.style.width = '50%';
            htmlPreview.style.width = '50%';
        }
        htmlPreview.style.display = 'block';
        htmlPreview.innerHTML =
marked((defaultRTE.contentModule.getEditPanel()).value);
        mdSource.parentElement.title = 'Code View';
    }
}
return (
    <RichTextEditorComponent id="markdownRTE"
        ref={ (richtexteditor) => { rteObj = richtexteditor! } }
        height='260px' editorMode='Markdown'
        formatter= {formatter}
        valueTemplate={template} toolbarSettings={toolbarSettings} >
        <Inject services={ [MarkdownEditor, Toolbar, Image, Link,
QuickToolbar] } />
    </RichTextEditorComponent>
);
}
export default App;
{% enddraw %}

```

File browser in React Rich text editor component

Rich Text Editor allows to browse and insert images in the edit panel using the file browser. File browser allows the users to browse and select a file or folder from the file system and it supports various cloud services.

Required additional dependency

The following list of additional dependencies are required to use the file browser feature in the Rich Text Editor.

```

`js
|-- @syncfusion/ej2-react-richtexteditor
|-- @syncfusion/ej2-layouts
|-- @syncfusion/ej2-grids
|-- @syncfusion/ej2-filemanager

```


Additional CSS Reference

Additionally add the below styles in the `[src/App.css]` file.

```
`css
@import "../../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-filemanager/styles/material.css";
`
```

The following example explains about how to configure the file browser within the Rich Text Editor component.

- Configure the `FileManager` toolbar item in the `toolbarSettings` API `items` property.
- Set `enable` property as `true` on `fileManagerSettings` property to make the file browser in the Rich Text Editor to appear on the `FileManager` toolbar click action.

Rich Text Editor features are segregated into individual feature-wise modules. To use the file browser tool, inject the `FileManager` module using services.

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - File Browser Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar, FileManager } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  fileManagerSettings = {
    enable: true,
    path: '/Pictures/Food',
    ajaxSettings: {
      url: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations',
      getImageUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage',
      uploadUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload',
      downloadUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download'
    }
  };
  toolbarSettings = {
    items: ['FileManager']
  };
  render() {
```

```

    return (<RichTextEditorComponent
fileManagerSettings={this.fileManagerSettings}
toolbarSettings={this.toolbarSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
    <p><b>Key features:</b></p>
    <ul>
    <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
    </li>
    <li>
        <p>Capable of handling markdown editing.</p>
    </li>
    <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
    </li>
    <li>
        <p>Provides a fully customizable toolbar.</p>
    </li>
    <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
    </li>
    <li>
        <p>Supports third-party library integration.</p>
    </li>
    <li>
        <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
    </li>
    <li>
        <p>Contains undo/redo manager.</p>
    </li>
    <li>
        <p>Creates bulleted and numbered lists.</p>
    </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
FileManager]}/>
    </RichTextEditorComponent>);
    }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - File Browser Sample
 */

```

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar, FileManager } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component<{}> {
  private fileManagerSettings: object = {
    enable: true,
    path: '/Pictures/Food',
    ajaxSettings: {
      url: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations',
      getImageUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage',
      uploadUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload',
      downloadUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download'
    }
  }
  private toolbarSettings: object = {
    items: ['FileManager']
  }
  public render() {
    return (
      <RichTextEditorComponent fileManagerSettings={this.fileManagerSettings}
toolbarSettings={this.toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
        </ul>
      </RichTextEditorComponent>
    );
  }
}

```

```

        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
    </li>
    <li>
        <p>Contains undo/redo manager.</p>
    </li>
    <li>
        <p>Creates bulleted and numbered lists.</p>
    </li>
</ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
FileManager]} />
</RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - File Browser Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar, FileManager } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
    let fileManagerSettings = {
        enable: true,
        path: '/Pictures/Food',
        ajaxSettings: {
            url: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations',
            getImageUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage',
            uploadUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload',
            downloadUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download'
        }
    };
    let toolbarSettings = {
        items: ['FileManager']
    };
    return (<RichTextEditorComponent
fileManagerSettings={fileManagerSettings} toolbarSettings={toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>

```

```

        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
FileManager]}/>
    </RichTextEditorComponent>;
  }
  export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - File Browser Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar, FileManager } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let fileManagerSettings: object = {
    enable: true,
    path: '/Pictures/Food',
    ajaxSettings: {
      url: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations',
      getImageUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage',

```

```

        uploadUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload',
        downloadUrl: 'https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download'
    }
}
let toolbarSettings: object = {
    items: ['FileManager']
}
return (
    <RichTextEditorComponent fileManagerSettings={fileManagerSettings}
toolbarSettings={toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
                <p>Capable of handling markdown editing.</p>
            </li>
            <li>
                <p>Contains a modular library to load the necessary functionality
on demand.</p>
            </li>
            <li>
                <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
                <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
                <p>Supports third-party library integration.</p>
            </li>
            <li>
                <p>Allows preview of modified content before saving it.</p>
            </li>
            <li>
                <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
            </li>
            <li>
                <p>Contains undo/redo manager.</p>
            </li>
            <li>
                <p>Creates bulleted and numbered lists.</p>
            </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
FileManager]} />
    </RichTextEditorComponent>
);
}
export default App;

```

Format Painter in React Rich Text Editor Component | Syncfusion

A format painter is a tool that allows you to copy the formatting from a piece of text and apply it to another one. Format Painter can be accessed via the toolbar or the keyboard shortcuts. The format painter can copy the formatting of a single word or a whole paragraph. The format painter can be customized using the [formatPainterSettings](#) property.

Enabling the toolbar option for Format Painter

You can add the `FormatPainter` tool in the Rich Text Editor using the `toolbarSettings.items` property.

Rich Text Editor features are segregated into individual feature-wise modules. To use the Format Painter feature, we need to import and inject the `FormatPainter` module in `services`.

By double-clicking the format painter toolbar button, sticky mode will be enabled. In sticky mode, the format painter will be disabled when the user clicks the `Escape` key again.

The following code example shows how to add the format painter tool in the Rich Text Editor.

[Class-component]

APP.JSX

```
import * as React from 'react';
import { RichTextEditorComponent, Inject, Toolbar, Link, Image, HtmlEditor,
QuickToolbar, FormatPainter } from '@syncfusion/ej2-react-richtexteditor';
class App extends React.Component {
  toolbarSettings = {
    items: ['FormatPainter', 'ClearFormat', 'Bold', 'Italic', 'Underline',
    '|', 'Formats', 'Alignments',
    'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', '|',
    'SourceCode', 'Undo', 'Redo']
  }
  render() {
    return (
      <RichTextEditorComponent toolbarSettings={this.toolbarSettings} >
        <h3>
          <strong>Format Painter</strong>
        </h3>
        <p>
          A Format Painter is a Rich Text Editor feature allowing users to
          quickly
          <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
            <strong>copy</strong>
          </span>
          and
          <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
            <strong>paste</strong>
          </span>
          formatting from one text to another. With a rich text editor,
          utilize the
          format painter as follows:
        </p>
        <ul>
          <li>Select the text whose format you want to copy.</li>
          <li>
            Click on the{" "}

```

```

        <strong>
          <em>Format Painter</em>
        </strong>{" "}
        button in the toolbar. It may look like a paintbrush icon.
      </li>
      <li>
        The cursor will change to a <strong>paintbrush</strong> icon.
Click and
        drag the cursor over the text you want to apply the copied
format.
      </li>
      <li>Release the mouse button to apply the format.</li>
    </ul>
    <p>
      Using the format painter in a rich text editor can save you time
when
      formatting a large document, You can quickly copy and apply
formatting to{" "}
      <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
        <strong>multiple sections</strong>
      </span>
      . It's a helpful tool for anyone who works with text editing
regularly, such
      as writers, editors, and content creators.
    </p>
    <Inject services={[ Toolbar, Link, Image, HtmlEditor, QuickToolbar,
FormatPainter ]} />
  </RichTextEditorComponent>
);
}
}
export default App;

```

APP.TSX

```

import * as React from 'react';
import { RichTextEditorComponent, Inject, Toolbar, Link, Image, HtmlEditor,
QuickToolbar, FormatPainter } from '@syncfusion/ej2-react-richtexteditor';
class App extends React.Component<{}, {}> {
  private toolbarSettings: object = {
    items: ['FormatPainter', 'ClearFormat', 'Bold', 'Italic', 'Underline',
    '|', 'Formats', 'Alignments',
    'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', '|',
    'SourceCode', 'Undo', 'Redo']
  }
  public render() {
    return (
      <RichTextEditorComponent toolbarSettings={this.toolbarSettings} >
        <h3>
          <strong>Format Painter</strong>
        </h3>
        <p>
          A Format Painter is a Rich Text Editor feature allowing users to
quickly
          <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
            <strong>copy</strong>
          </span>

```



```

        </span>
        and
        <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
          <strong>paste</strong>
        </span>
        formatting from one text to another. With a rich text editor,
utilize the
        format painter as follows:
      </p>
      <ul>
        <li>Select the text whose format you want to copy.</li>
        <li>
          Click on the{" "}
          <strong>
            <em>Format Painter</em>
          </strong>{" "}
          button in the toolbar. It may look like a paintbrush icon.
        </li>
        <li>
          The cursor will change to a <strong>paintbrush</strong> icon.
Click and
          drag the cursor over the text you want to apply the copied
format.
        </li>
        <li>Release the mouse button to apply the format.</li>
      </ul>
      <p>
        Using the format painter in a rich text editor can save you time
when
        formatting a large document, You can quickly copy and apply
formatting to{" "}
        <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
          <strong>multiple sections</strong>
        </span>
        . It's a helpful tool for anyone who works with text editing
regularly, such
        as writers, editors, and content creators.
      </p>
      <Inject services={[ Toolbar, Link, Image, HtmlEditor, QuickToolbar,
FormatPainter ]} />
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

import * as React from 'react';
import { RichTextEditorComponent, Inject, Toolbar, Link, Image, HtmlEditor,
QuickToolbar, FormatPainter } from '@syncfusion/ej2-react-richtexteditor';
function App() {
  let toolbarSettings = {

```

```

        items: ['FormatPainter', 'ClearFormat', 'Bold', 'Italic', 'Underline',
        '|', 'Formats', 'Alignments',
        'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', '|',
        'SourceCode', 'Undo', 'Redo']
    }
    return (
        <RichTextEditorComponent toolbarSettings={toolbarSettings} >
            <h3>
                <strong>Format Painter</strong>
            </h3>
            <p>
                A Format Painter is a Rich Text Editor feature allowing users to
                quickly
                <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
                    <strong>copy</strong>
                </span>
                and
                <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
                    <strong>paste</strong>
                </span>
                formatting from one text to another. With a rich text editor,
                utilize the
                format painter as follows:
            </p>
            <ul>
                <li>Select the text whose format you want to copy.</li>
                <li>
                    Click on the{" "}
                    <strong>
                        <em>Format Painter</em>
                    </strong>{" "}
                    button in the toolbar. It may look like a paintbrush icon.
                </li>
                <li>
                    The cursor will change to a <strong>paintbrush</strong> icon.
                    Click and
                    drag the cursor over the text you want to apply the copied
                    format.
                </li>
                <li>Release the mouse button to apply the format.</li>
            </ul>
            <p>
                Using the format painter in a rich text editor can save you time
                when
                formatting a large document, You can quickly copy and apply
                formatting to{" "}
                <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
                    <strong>multiple sections</strong>
                </span>
                . It's a helpful tool for anyone who works with text editing
                regularly, such
                as writers, editors, and content creators.
            </p>
            <Inject services={[ Toolbar, Link, Image, HtmlEditor, QuickToolbar,
            FormatPainter ]} />
        </RichTextEditorComponent>
    );

```

```
}
export default App;
```

APP.TSX

```
import * as React from 'react';
import { RichTextEditorComponent, Inject, Toolbar, Link, Image, HtmlEditor,
QuickToolbar, FormatPainter } from '@syncfusion/ej2-react-richtexteditor';
function App() {
  let toolbarSettings: object = {
    items: ['FormatPainter', 'ClearFormat', 'Bold', 'Italic', 'Underline',
    '|', 'Formats', 'Alignments',
    'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', '|',
    'SourceCode', 'Undo', 'Redo']
  }
  return (
    <RichTextEditorComponent toolbarSettings={toolbarSettings} >
      <h3>
        <strong>Format Painter</strong>
      </h3>
      <p>
        A Format Painter is a Rich Text Editor feature allowing users to
        quickly
        <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
          <strong>copy</strong>
        </span>
        and
        <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
          <strong>paste</strong>
        </span>
        formatting from one text to another. With a rich text editor,
        utilize the
        format painter as follows:
      </p>
      <ul>
        <li>Select the text whose format you want to copy.</li>
        <li>
          Click on the{" "}
          <strong>
            <em>Format Painter</em>
          </strong>{" "}
          button in the toolbar. It may look like a paintbrush icon.
        </li>
        <li>
          The cursor will change to a <strong>paintbrush</strong> icon.
          Click and
          drag the cursor over the text you want to apply the copied
          format.
        </li>
        <li>Release the mouse button to apply the format.</li>
      </ul>
      <p>
        Using the format painter in a rich text editor can save you time
        when
        formatting a large document, You can quickly copy and apply
        formatting to{" "}

```

```

    <span style={{ backgroundColor: "rgb(198, 140, 83)" }}>
      <strong>multiple sections</strong>
    </span>
    . It's a helpful tool for anyone who works with text editing
    regularly, such
      as writers, editors, and content creators.
    </p>
    <Inject services={[ Toolbar, Link, Image, HtmlEditor, QuickToolbar,
    FormatPainter ]} />
  </RichTextEditorComponent>
);
}
export default App;
```

Customization of copy and paste format

You can customize the format painter tool in the Rich Text Editor using the `formatPainterSettings` property.

The `allowedFormats` property helps you to specify tag names that allow the formats to be copied from the selected text. For instance, you can include formats from the selected text using tags like `p`; `h1`; `h2`; `h3`; `div`; `ul`; `ol`; `li`; `span`; `strong`; `em`; `code`. The following example demonstrates how to customize this functionality.

Similarly, with the `deniedFormats` property, you can utilize the selectors to prevent specific formats from being pasted onto the selected text. The table below illustrates the selectors and their respective usage.

Type	Description	Selector	Usage
----- ----- ----- -----			
()	Class Selector	h3(e-rte-block-blue-text)	The class name e-rte-block-blue-text of H3 element is not copied.
[]	Attribute Selector	span\[title\]	The title attribute of span element is not copied.
{}	Style Selector	span{background-color, color}	The background-color and color styles of span element is not copied.

Using the `deniedFormats` property following styles are denied copying from the selected text such as `h3(e-rte-block-blue-text){background-color,padding}[title]`; `li{color}`; `span(e-inline-text-highlight)[title]`; `strong{color}(e-rte-strong-bg)`.

[Class-component]

APP.JSX

```

import * as React from 'react';
import { RichTextEditorComponent, Inject, Toolbar, Link, Image, HtmlEditor,
QuickToolbar, FormatPainter } from '@syncfusion/ej2-react-richtexteditor';
class App extends React.Component {
  toolbarSettings = {
```

```

        items: ['FormatPainter', 'ClearFormat', 'Bold', 'Italic', 'Underline',
        '|', 'Formats', 'Alignments',
        'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', '|',
        'SourceCode', 'Undo', 'Redo']
    }
    formatPainterSettings = {
        allowedFormats: 'p;h1;h2;h3;div;ul;ol;li;span;strong;em;code;',
        deniedFormats: 'h3(e-rte-block-blue-text){background-
color,padding,color}[title]; li{color}; span(e-inline-text-
highlight){color}[title]; strong{color}(e-rte-strong-bg);',
    }
    render() {
        return (
            <RichTextEditorComponent toolbarSettings={this.toolbarSettings}
formatPainterSettings={this.formatPainterSettings} >
                <h3 className="e-rte-block-blue-text" title="Format Painter"
style={{ color: "#0079f3", backgroundColor: "#eff6ff", padding: 10 }}>
                    <strong>Format Painter</strong>
                </h3>
                <p>
                    A Format Painter is a Rich Text Editor feature allowing users to
quickly
                    <span className="e-inline-text-highlight" style={{ color: "blue"
}} title="Styled by CSS Class selector">
                        <strong>copy</strong>
                    </span>
                    and
                    <span className="e-inline-text-highlight" style={{ color: "blue"
}} title="Styled by CSS Class selector">
                        <strong>paste</strong>
                    </span>
                    formatting from one text to another. With a rich text editor,
utilize the
                    format painter as follows:
                </p>
                <ul>
                    <li style={{ color: "crimson" }}>
                        Select the text whose format you want to copy.
                    </li>
                    <li style={{ color: "crimson" }}>
                        Click on the{" "}
                        <strong>
                            <em>Format Painter</em>
                        </strong>{" "}
                        button in the toolbar. It may look like a paintbrush icon.
                    </li>
                    <li style={{ color: "crimson" }}>
                        The cursor will change to a <strong>paintbrush</strong> icon.
Click and
                        drag the cursor over the text you want to apply the copied
format.
                    </li>
                    <li style={{ color: "crimson" }}>
                        Release the mouse button to apply the format.
                    </li>
                </ul>
                <p>

```

```

        Using the format painter in a rich text editor can save you time
when
        formatting a large document, You can quickly copy and apply
formatting to{" "}
        <strong className="e-rte-strong-bg" style={{ color: "blue" }}>
            multiple sections
        </strong>
        . It's a helpful tool for anyone who works with text editing
regularly, such
        as writers, editors, and content creators.
    </p>
    <Inject services={[ Toolbar, Link, Image, HtmlEditor, QuickToolbar,
FormatPainter ]} />
    </RichTextEditorComponent>
    );
    }
}
export default App;

```

APP.TSX

```

import * as React from 'react';
import { RichTextEditorComponent, Inject, Toolbar, Link, Image, HtmlEditor,
QuickToolbar, FormatPainter, FormatPainterSettingsModel } from
'@syncfusion/ej2-react-richtexteditor';
class App extends React.Component<{}, {}> {
    private toolbarSettings: object = {
        items: ['FormatPainter', 'ClearFormat', 'Bold', 'Italic', 'Underline',
'|', 'Formats', 'Alignments',
        'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', '|',
'SourceCode', 'Undo', 'Redo']
    }
    private formatPainterSettings: FormatPainterSettingsModel = {
        allowedFormats: 'p;h1;h2;h3;div;ul;ol;li;span;strong;em;code;',
        deniedFormats: 'h3(e-rte-block-blue-text){background-
color,padding,color}[title]; li{color}; span(e-inline-text-
highlight){color}[title]; strong{color}(e-rte-strong-bg);',
    }
    public render() {
        return (
            <RichTextEditorComponent toolbarSettings={this.toolbarSettings}
formatPainterSettings={this.formatPainterSettings} >
                <h3 className="e-rte-block-blue-text" title="Format Painter"
style={{ color: "#0079f3", backgroundColor: "#eff6ff", padding: 10 }}>
                    <strong>Format Painter</strong>
                </h3>
                <p>
                    A Format Painter is a Rich Text Editor feature allowing users to
quickly
                    <span className="e-inline-text-highlight" style={{ color: "blue"
}} title="Styled by CSS Class selector">
                        <strong>copy</strong>
                    </span>
                    and
                    <span className="e-inline-text-highlight" style={{ color: "blue"
}} title="Styled by CSS Class selector">

```

```

        <strong>paste</strong>
      </span>
      formatting from one text to another. With a rich text editor,
utilize the
      format painter as follows:
    </p>
    <ul>
      <li style={{ color: "crimson" }}>
        Select the text whose format you want to copy.
      </li>
      <li style={{ color: "crimson" }}>
        Click on the{" "}
        <strong>
          <em>Format Painter</em>
        </strong>{" "}
        button in the toolbar. It may look like a paintbrush icon.
      </li>
      <li style={{ color: "crimson" }}>
        The cursor will change to a <strong>paintbrush</strong> icon.
Click and
        drag the cursor over the text you want to apply the copied
format.
      </li>
      <li style={{ color: "crimson" }}>
        Release the mouse button to apply the format.
      </li>
    </ul>
    <p>
      Using the format painter in a rich text editor can save you time
when
      formatting a large document, You can quickly copy and apply
formatting to{" "}
      <strong className="e-rte-strong-bg" style={{ color: "blue" }}>
        multiple sections
      </strong>
      . It's a helpful tool for anyone who works with text editing
regularly, such
      as writers, editors, and content creators.
    </p>
    <Inject services={[ Toolbar, Link, Image, HtmlEditor, QuickToolbar,
FormatPainter ]} />
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

import * as React from 'react';
import { RichTextEditorComponent, Inject, Toolbar, Link, Image, HtmlEditor,
QuickToolbar, FormatPainter } from '@syncfusion/ej2-react-richtexteditor';
function App() {
  let toolbarSettings = {

```

```

    items: ['FormatPainter', 'ClearFormat', 'Bold', 'Italic', 'Underline',
    '|', 'Formats', 'Alignments',
    'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', '|',
    'SourceCode', 'Undo', 'Redo']
  }
  let formatPainterSettings = {
    allowedFormats: 'p;h1;h2;h3;div;ul;ol;li;span;strong;em;code;',
    deniedFormats: 'h3(e-rte-block-blue-text){background-
color,padding,color}[title]; li{color}; span(e-inline-text-
highlight){color}[title]; strong{color}(e-rte-strong-bg);',
  }
  return (
    <RichTextEditorComponent toolbarSettings={toolbarSettings}
formatPainterSettings={formatPainterSettings} >
      <h3 className="e-rte-block-blue-text" title="Format Painter" style={{
color: "#0079f3", backgroundColor: "#eff6ff", padding: 10 }}>
        <strong>Format Painter</strong>
      </h3>
      <p>
        A Format Painter is a Rich Text Editor feature allowing users to
quickly
        <span className="e-inline-text-highlight" style={{ color: "blue" }}
title="Styled by CSS Class selector">
          <strong>copy</strong>
        </span>
        and
        <span className="e-inline-text-highlight" style={{ color: "blue" }}
title="Styled by CSS Class selector">
          <strong>paste</strong>
        </span>
        formatting from one text to another. With a rich text editor,
utilize the
        format painter as follows:
      </p>
      <ul>
        <li style={{ color: "crimson" }}>
          Select the text whose format you want to copy.
        </li>
        <li style={{ color: "crimson" }}>
          Click on the{" "}
          <strong>
            <em>Format Painter</em>
          </strong>{" "}
          button in the toolbar. It may look like a paintbrush icon.
        </li>
        <li style={{ color: "crimson" }}>
          The cursor will change to a <strong>paintbrush</strong> icon.
Click and
          drag the cursor over the text you want to apply the copied
format.
        </li>
        <li style={{ color: "crimson" }}>
          Release the mouse button to apply the format.
        </li>
      </ul>
    </p>
  )

```



```

        Using the format painter in a rich text editor can save you time
when
        formatting a large document, You can quickly copy and apply
formatting to{" "}
        <strong className="e-rte-strong-bg" style={{ color: "blue" }}>
            multiple sections
        </strong>
        . It's a helpful tool for anyone who works with text editing
regularly, such
        as writers, editors, and content creators.
    </p>
    <Inject services={[ Toolbar, Link, Image, HtmlEditor, QuickToolbar,
FormatPainter ]} />
    </RichTextEditorComponent>
  );
}
export default App;

```

APP.TSX

```

import * as React from 'react';
import { RichTextEditorComponent, Inject, Toolbar, Link, Image, HtmlEditor,
QuickToolbar, FormatPainter, FormatPainterSettingsModel } from
'@syncfusion/ej2-react-richtexteditor';
function App() {
  let toolbarSettings: object = {
    items: ['FormatPainter', 'ClearFormat', 'Bold', 'Italic', 'Underline',
'|', 'Formats', 'Alignments',
'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', '|',
'SourceCode', 'Undo', 'Redo']
  }
  let formatPainterSettings: FormatPainterSettingsModel = {
    allowedFormats: 'p;h1;h2;h3;div;ul;ol;li;span;strong;em;code;',
    deniedFormats: 'h3(e-rte-block-blue-text){background-
color,padding,color}[title]; li{color}; span(e-inline-text-
highlight){color}[title]; strong{color}(e-rte-strong-bg);',
  }
  return (
    <RichTextEditorComponent toolbarSettings={toolbarSettings}
formatPainterSettings={formatPainterSettings} >
      <h3 className="e-rte-block-blue-text" title="Format Painter" style={{
color: "#0079f3", backgroundColor: "#eff6ff", padding: 10 }}>
        <strong>Format Painter</strong>
      </h3>
      <p>
        A Format Painter is a Rich Text Editor feature allowing users to
quickly
        <span className="e-inline-text-highlight" style={{ color: "blue" }}
title="Styled by CSS Class selector">
          <strong>copy</strong>
        </span>
        and
        <span className="e-inline-text-highlight" style={{ color: "blue" }}
title="Styled by CSS Class selector">
          <strong>paste</strong>
        </span>

```

```

        formatting from one text to another. With a rich text editor,
utilize the
        format painter as follows:
    </p>
    <ul>
        <li style={{ color: "crimson" }}>
            Select the text whose format you want to copy.
        </li>
        <li style={{ color: "crimson" }}>
            Click on the{" "}
            <strong>
                <em>Format Painter</em>
            </strong>{" "}
            button in the toolbar. It may look like a paintbrush icon.
        </li>
        <li style={{ color: "crimson" }}>
            The cursor will change to a <strong>paintbrush</strong> icon.
Click and
            drag the cursor over the text you want to apply the copied
format.
        </li>
        <li style={{ color: "crimson" }}>
            Release the mouse button to apply the format.
        </li>
    </ul>
    <p>
        Using the format painter in a rich text editor can save you time
when
        formatting a large document, You can quickly copy and apply
formatting to{" "}
        <strong className="e-rte-strong-bg" style={{ color: "blue" }}>
            multiple sections
        </strong>
        . It's a helpful tool for anyone who works with text editing
regularly, such
        as writers, editors, and content creators.
    </p>
    <Inject services={[ Toolbar, Link, Image, HtmlEditor, QuickToolbar,
FormatPainter ]} />
    </RichTextEditorComponent>
    );
}
export default App;

```

Using the shortcut key to copy and paste the format

You can use the following shortcut keys to copy and paste the format in the Rich Text Editor.

Actions	Keyboard shortcuts	Description
-----	-----	-----
Copy the format	Alt + Shift + c	Copy the selection format or current range.
Pate the format	Alt + Shift + v	Paint the copied format.

Escape the sticky mode.	Esc	Remove the previously copied format and disable
----------------------------	-----	---

The format painter retains the formatting after application making it possible to apply the same formatting multiple times by using the Alt + Shift + v keyboard shortcut.

Additionally, You can perform the format painter actions programmatically using the [executeCommand](#) public method.

Form support in React Rich text editor component

The following sample demonstrates how to get the Rich Text Editor value in button click.

Render the Rich Text Editor in form.

```
`ts
<form id="myForm" class="form-vertical">
  <div class="form-group">
    <textarea id="defaultRTE" name="defaultRTE" required maxlength="100" minlength="20" data-msg-
    containerid="dateError"></textarea>
    <div id="dateError"></div>
  </div>
  <div class="form-btn-section">
    <button id="validateSubmit" class="sample-btn e-control e-btn" type="submit" data-
    ripple="true">Submit</button>
    <button id="resetbtn" class="sample-btn e-control e-btn" type="reset" data-
    ripple="true">Reset</button>
  </div>
</form>
`
```

Upon submitting the form, the `getValue` method will be triggered. Through the `FormData` class, get the Rich Text Editor value.

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - Form Sample
 */
import { FormValidator } from '@syncfusion/ej2-inputs';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
// import * as Marked from 'marked';
import * as React from 'react';
class App extends React.Component {
  formObject;
  componentDidMount() {
```

```

        const option = {
            rules: {
                defaultRTE: {
                    maxLength: [108, 'Maximum 100 character only'],
                    minLength: [15, 'Need atleast 8 character length'],
                    required: true,
                }
            }
        };
        this.formObject = new FormValidator('#myForm', option);
        document.getElementById('validateSubmit').addEventListener('click',
(e) => {
            const form = document.forms.myForm;
            const formData = new FormData(form);
            const rteValue = formData.get('defaultRTE');
            // Use this value to the data base.
            alert(rteValue);
            e.preventDefault();
        });
    }
    render() {
        return (<form id="myForm" className="form-vertical">
            <div className="form-group">
                <RichTextEditorComponent id="defaultRTE" name="defaultRTE"
className="form-control" height={200} showCharCount={true} maxLength={100}
placeholder={'Type something'} value={''}>
                <Inject services={[Toolbar, Image, Link, HtmlEditor,
QuickToolbar, Count]}/>
            </RichTextEditorComponent>
            </div>
            <div className="form-btn-section">
                <button id="validateSubmit" className="sample-btn e-control e-btn"
type="submit" data-ripple="true">Submit</button>
                <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
            </div>
        </form>);
    }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Form Sample
 */
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
// import * as Marked from 'marked';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    public formObject: FormValidatorModel;
    public componentDidMount(): void {
        const option: FormValidatorModel = {

```

```

    rules: {
      defaultRTE: {
        maxLength:[108, 'Maximum 100 character only'],
        minLength: [15, 'Need atleast 8 character length'],
        required: true,
      }
    }
  };
  this.formObject = new FormValidator('#myForm', option);
  (document as any).getElementById('validateSubmit').addEventListener('click', (e: any) => {
    const form = (document.forms as any).myForm;
    const formData = new FormData(form);
    const rteValue = formData.get('defaultRTE');
    // Use this value to the data base.
    alert(rteValue);
    e.preventDefault();
  });
}
public render() {
  return (
    <form id="myForm" className="form-vertical">
      <div className="form-group">
        <RichTextEditorComponent id="defaultRTE" name="defaultRTE"
className="form-control" height={200} showCharCount={true} maxLength={100}
placeholder={'Type something'} value={''}>
          <Inject services={[Toolbar, Image, Link, HtmlEditor,
QuickToolbar, Count]} />
        </RichTextEditorComponent>
      </div>
      <div className="form-btn-section">
        <button id="validateSubmit" className="sample-btn e-control e-btn"
type="submit" data-ripple="true">Submit</button>
        <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
      </div>
    </form>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Form Sample
 */
import { FormValidator } from '@syncfusion/ej2-inputs';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
// import * as Marked from 'marked';
import * as React from 'react';
function App() {

```

```

let formObject;
function componentCreated() {
  const option = {
    rules: {
      defaultRTE: {
        maxLength: [108, 'Maximum 100 character only'],
        minLength: [15, 'Need atleast 8 character length'],
        required: true,
      }
    }
  };
  formObject = new FormValidator('#myForm', option);
  document.getElementById('validateSubmit').addEventListener('click',
(e) => {
    const form = document.forms.myForm;
    const formData = new FormData(form);
    const rteValue = formData.get('defaultRTE');
    // Use this value to the data base.
    alert(rteValue);
    e.preventDefault();
  });
}
return (<form id="myForm" className="form-vertical">
  <div className="form-group">
    <RichTextEditorComponent id="defaultRTE" name="defaultRTE"
created={componentCreated} className="form-control" height={200}
showCharCount={true} maxLength={100} placeholder={'Type something'}
value={' '}>
    <Inject services={[Toolbar, Image, Link, HtmlEditor,
QuickToolbar, Count]}/>
    </RichTextEditorComponent>
  </div>
  <div className="form-btn-section">
    <button id="validateSubmit" className="sample-btn e-control e-btn"
type="submit" data-ripple="true">Submit</button>
    <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
  </div>
</form>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Form Sample
 */
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
// import * as Marked from 'marked';
import * as React from 'react';
function App() {
  let formObject: FormValidatorModel;
  function componentCreated(): void {

```

```

const option: FormValidatorModel = {
  rules: {
    defaultRTE: {
      maxLength:[108, 'Maximum 100 character only'],
      minLength: [15, 'Need atleast 8 character length'],
      required: true,
    }
  }
};
formObject = new FormValidator('#myForm', option);
(document as any).getElementById('validateSubmit').addEventListener('click', (e: any) => {
  const form = (document.forms as any).myForm;
  const formData = new FormData(form);
  const rteValue = formData.get('defaultRTE');
  // Use this value to the data base.
  alert(rteValue);
  e.preventDefault();
});
}
return (
  <form id="myForm" className="form-vertical">
    <div className="form-group">
      <RichTextEditorComponent id="defaultRTE" created={componentCreated}
name="defaultRTE" className="form-control" height={200} showCharCount={true}
maxLength={100} placeholder={'Type something'} value={''}>
        <Inject services={[Toolbar, Image, Link, HtmlEditor,
QuickToolbar, Count]} />
      </RichTextEditorComponent>
    </div>
    <div className="form-btn-section">
      <button id="validateSubmit" className="sample-btn e-control e-btn"
type="submit" data-ripple="true">Submit</button>
      <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
    </div>
  </form>
);
}
export default App;

```

Validation in React Rich text editor component

Validation rules

The Rich Text Editor provides the functionality of character count and its validation. So, you can validate the Rich Text Editor's value on form submission by applying validationRules and validationMessage to the Rich Text Editor.

| Rules | Description |

| --- | --- |

| required | Requires the value for the Rich Text Editor control. |

| minlength | Requires the value to be of given minimum characters count. |

| maxlength | Requires the value to be of given maximum characters count. |

This sample is used to validate form using the obtrusive Validation. Type the values in Rich Text Editor and the form enables the validation with the formvalidator rules by clicking on the submit externally. All rules are validated by the formvalidator rules.

Validation message

The default error message for a rule can be customizable by defining it along with the concern rule object as follows.

In the following sample, customize the error message along with the concern rule.

[Class-component]

APP.JSX

```
{% raw %}
/**
 * Rich Text Editor - Form Validation Sample
 */
import { FormValidator } from '@syncfusion/ej2-inputs';
import { HtmlEditor, Image, Count, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
class App extends React.Component {
  formObject;
  button;
  componentDidMount() {
    const option = {
      rules: {
        'defaultRTE-value': {
          maxLength: [108, 'Maximum 100 character only'],
          minLength: [15, 'Need atleast 8 character length'],
          required: true
        }
      }
    };
    this.formObject = new FormValidator('#myForm', option);
    document.getElementById('validateSubmit').addEventListener('click',
(e) => {
      const form = document.forms.myForm;
      const formData = new FormData(form);
      const rteValue = formData.get('defaultRTE-value');
      // Use this value to the data base.
      alert(rteValue);
      e.preventDefault();
    });
  }
  onChange() {
    this.button.disabled = false;
  }
  render() {
    return (<form id="myForm" className="form-vertical">
      <div className="form-group">
        <RichTextEditorComponent id="defaultRTE" htmlAttributes={{ name:
"defaultRTE-value" }} className="form-control" height={200}

```



```

showCharCount={true} maxLength={100} placeholder={'Type something'}
change={this.onChange.bind(this)} value={' '}>
    <Inject services={[Toolbar, Image, Link, Count, HtmlEditor,
QuickToolbar]}/>
    </RichTextEditorComponent>
</div>
<div className="form-btn-section">
    <ButtonComponent id="validateSubmit" ref={(scope) => { this.button
= scope; }} disabled={true}>Submit</ButtonComponent>
    <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
</div>
</form>;
}
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Form Validation Sample
 */
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
import { HtmlEditor, Image, Count, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    public formObject: FormValidator;
    public button: ButtonComponent;
    public componentDidMount(): void {
        const option: FormValidatorModel = {
            rules: {
                'defaultRTE-value': {
                    maxLength: [108, 'Maximum 100 character only'],
                    minLength: [15, 'Need atleast 8 character length'],
                    required: true
                }
            }
        };
        this.formObject = new FormValidator('#myForm', option);
        (document as
any).getElementById('validateSubmit').addEventListener('click', (e: any) => {
            const form = (document.forms as any).myForm;
            const formData = new FormData(form);
            const rteValue = formData.get('defaultRTE-value');
            // Use this value to the data base.
            alert(rteValue);
            e.preventDefault();
        });
    }
    private onChange(): void {
        this.button.disabled = false;
    }
}

```

```

    }
    public render() {
      return (
        <form id="myForm" className="form-vertical">
          <div className="form-group">
            <RichTextEditorComponent id="defaultRTE" htmlAttributes= {{ name:
"defaultRTE-value" }} className="form-control" height={200}
showCharCount={true} maxLength={100} placeholder='Type something'
change={this.onChange.bind(this)} value=''>
              <Inject services={[Toolbar, Image, Link, Count, HtmlEditor,
QuickToolbar]} />
            </RichTextEditorComponent>
          </div>
          <div className="form-btn-section">
            <ButtonComponent id="validateSubmit" ref={(scope) => { this.button
= scope; }} disabled={true}>Submit</ButtonComponent>
            <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
          </div>
        </form>
      );
    }
  }
  export default App;
  {% endraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - Form Validation Sample
 */
import { FormValidator } from '@syncfusion/ej2-inputs';
import { HtmlEditor, Image, Count, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import { useEffect } from "react";
function App() {
  useEffect(() => {
    const option = {
      rules: {
        'defaultRTE-value': {
          maxLength: [108, 'Maximum 100 character only'],
          minLength: [15, 'Need atleast 8 character length'],
          required: true
        }
      }
    };
    formObject = new FormValidator('#myForm', option);
    document.getElementById('validateSubmit').addEventListener('click',
(e) => {
      const form = document.forms.myForm;

```

```

        const formData = new FormData(form);
        const rteValue = formData.get('defaultRTE-value');
        // Use this value to the data base.
        alert(rteValue);
        e.preventDefault();
    });
});
let button;
let formObject;
function onChange() {
    button.disabled = false;
}
return (<form id="myForm" className="form-vertical">
    <div className="form-group">
        <RichTextEditorComponent id="defaultRTE" htmlAttributes={{ name:
"defaultRTE-value" }} className="form-control" height={200}
showCharCount={true} maxLength={100} placeholder={'Type something'}
change={onChange.bind(this)} value={' '}>
            <Inject services={[Toolbar, Image, Link, Count, HtmlEditor,
QuickToolbar]}/>
        </RichTextEditorComponent>
    </div>
    <div className="form-btn-section">
        <ButtonComponent id="validateSubmit" ref={(scope) => { button =
scope; }} disabled={true}>Submit</ButtonComponent>
        <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
    </div>
</form>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Form Validation Sample
 */
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
import { HtmlEditor, Image, Count, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import { useEffect } from "react";
function App() {
    useEffect(() => {
        const option: FormValidatorModel = {
            rules: {
                'defaultRTE-value': {
                    maxLength: [108, 'Maximum 100 character only'],
                    minLength: [15, 'Need atleast 8 character length'],
                    required: true
                }
            }
        }
    })
}
{% endraw %}

```

```

    };
    formObject = new FormValidator('#myForm', option);
    (document as
any).getElementById('validateSubmit').addEventListener('click', (e: any) => {
    const form = (document.forms as any).myForm;
    const formData = new FormData(form);
    const rteValue = formData.get('defaultRTE-value');
    // Use this value to the data base.
    alert(rteValue);
    e.preventDefault();
    });
});
let button: ButtonComponent;
let formObject: FormValidator;
function onChange(): void {
    button.disabled = false;
}
return (
    <form id="myForm" className="form-vertical">
        <div className="form-group">
            <RichTextEditorComponent id="defaultRTE" htmlAttributes= {{ name:
"defaultRTE-value" }} className="form-control" height={200}
showCharCount={true} maxLength={100} placeholder={'Type something'}
change={onChange.bind(this)} value={' '}>
                <Inject services={[Toolbar, Image, Link, Count, HtmlEditor,
QuickToolbar]} />
            </RichTextEditorComponent>
        </div>
        <div className="form-btn-section">
            <ButtonComponent id="validateSubmit" ref={(scope) => { button =
scope; }} disabled={true}>Submit</ButtonComponent>
            <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
        </div>
    </form>
);
}
export default App;
{% endraw %}

```

Custom placement of validation message

The FormValidator has an event customPlacement which can be used to place the error message from default position to desired custom location.

[Class-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - Custom Placement of Validation Message Sample
 */
import { FormValidator } from '@syncfusion/ej2-inputs';
import { HtmlEditor, Image, Inject, Count, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';

```

```

import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
class App extends React.Component {
  formObject;
  button;
  componentDidMount() {
    const option = {
      customPlacement: (inputElement, error) => {
        // Initialize the CustomPlacement.
        document.getElementById('data-error').appendChild(error);
      },
      rules: {
        'defaultRTE-value': {
          maxLength: [108, 'RTE: Maximum 100 character only'],
          minLength: [15, 'RTE: Need atleast 8 character length'],
          required: [true, 'RTE: value is required']
        }
      }
    };
    this.formObject = new FormValidator('#myForm', option);
    document.getElementById('validateSubmit').addEventListener('click',
(e) => {
      const form = document.forms.myForm;
      const formData = new FormData(form);
      const rteValue = formData.get('defaultRTE-value');
      // Use this value to the data base.
      alert(rteValue);
      e.preventDefault();
    });
  }
  onChange() {
    this.button.disabled = false;
  }
  render() {
    return (<form id="myForm" className="form-vertical">
      <div className="form-group">
        <RichTextEditorComponent id="defaultRTE" htmlAttributes={{ name:
"defaultRTE-value" }} className="form-control" height={200}
showCharCount={true} maxLength={100} placeholder={'Type something'}
change={this.onChange.bind(this)} value={''}>
          <Inject services={[Toolbar, Image, Link, Count, HtmlEditor,
QuickToolbar]}/>
        </RichTextEditorComponent>
      </div>
      <div className="form-group">
        <div className="col-sm-3 control-label"/>
        <div className="col-sm-6">
          <div id='data-error' />
        </div>
      </div>
      <div className="form-btn-section">
        <ButtonComponent id="validateSubmit" ref={(scope) => {
this.button = scope; }} disabled={true}>Submit</ButtonComponent>
        <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
      </div>
    </form>);
  }
}

```

```

    }
  }
  export default App;
  {% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Custom Placement of Validation Message Sample
 */
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
import { HtmlEditor, Image, Inject, Count, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  public formObject: FormValidator;
  public button: ButtonComponent;
  public componentDidMount(): void {
    const option: FormValidatorModel = {
      customPlacement: (inputElement: HTMLElement, error: HTMLElement) => {
        // Initialize the CustomPlacement.
        (document as any).getElementById('data-error').appendChild(error);
      },
      rules: {
        'defaultRTE-value': {
          maxLength: [108, 'RTE: Maximum 100 character only'],
          minLength: [15, 'RTE: Need atleast 8 character length'],
          required: [true, 'RTE: value is required']
        }
      }
    };
    this.formObject = new FormValidator('#myForm', option);
    (document as any).getElementById('validateSubmit').addEventListener('click', (e: any) => {
      const form = (document as any).forms.myForm;
      const formData = new FormData(form);
      const rteValue = formData.get('defaultRTE-value');
      // Use this value to the data base.
      alert(rteValue);
      e.preventDefault();
    });
  }
  private onChange(): void {
    this.button.disabled = false;
  }
  public render() {
    return (
      <form id="myForm" className="form-vertical">
        <div className="form-group">
          <RichTextEditorComponent id="defaultRTE" htmlAttributes= {{ name:
"defaultRTE-value" }} className="form-control" height={200}
showCharCount={true} maxLength={100} placeholder='Type something'
change={this.onChange.bind(this)} value=''>

```

```

        <Inject services={[Toolbar, Image, Link, Count, HtmlEditor,
QuickToolbar]} />
      </RichTextEditorComponent>
    </div>
    <div className="form-group">
      <div className="col-sm-3 control-label"/>
      <div className="col-sm-6">
        <div id='data-error' />
      </div>
    </div>
    <div className="form-btn-section">
      <ButtonComponent id="validateSubmit" ref={(scope) => {
this.button = scope; }} disabled={true}>Submit</ButtonComponent>
      <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
    </div>
  </form>
);
}
}
export default App;
{% enddraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - Custom Placement of Validation Message Sample
 */
import { FormValidator } from '@syncfusion/ej2-inputs';
import { HtmlEditor, Image, Inject, Count, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import { useEffect } from "react";
function App() {
  useEffect(() => {
    const option = {
      customPlacement: (inputElement, error) => {
        // Initialize the CustomPlacement.
        document.getElementById('data-error').appendChild(error);
      },
      rules: {
        'defaultRTE-value': {
          maxLength: [108, 'RTE: Maximum 100 character only'],
          minLength: [15, 'RTE: Need atleast 8 character length'],
          required: [true, 'RTE: value is required']
        }
      }
    };
    formObject = new FormValidator('#myForm', option);
    document.getElementById('validateSubmit').addEventListener('click',
(e) => {

```

```

        const form = document.forms.myForm;
        const formData = new FormData(form);
        const rteValue = formData.get('defaultRTE-value');
        // Use this value to the data base.
        alert(rteValue);
        e.preventDefault();
    });
});
let formObject;
let button;
function onChange() {
    button.disabled = false;
}
return (<form id="myForm" className="form-vertical">
    <div className="form-group">
        <RichTextEditorComponent id="defaultRTE" htmlAttributes={{ name:
"defaultRTE-value" }} className="form-control" height={200}
showCharCount={true} maxLength={100} placeholder='Type something'
change={onChange.bind(this)} value={' '}>
            <Inject services={[Toolbar, Image, Link, Count, HtmlEditor,
QuickToolbar]}/>
        </RichTextEditorComponent>
    </div>
    <div className="form-group">
        <div className="col-sm-3 control-label"/>
        <div className="col-sm-6">
            <div id='data-error' />
        </div>
    </div>
    <div className="form-btn-section">
        <ButtonComponent id="validateSubmit" ref={(scope) => { button =
scope; }} disabled={true}>Submit</ButtonComponent>
        <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
    </div>
</form>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Custom Placement of Validation Message Sample
 */
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
import { HtmlEditor, Image, Inject, Count, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import * as React from 'react';
import { useEffect } from "react";
function App() {
    useEffect(() => {
        const option: FormValidatorModel = {

```



```

    customPlacement: (inputElement: HTMLElement, error: HTMLElement)=> {
      // Initialize the CustomPlacement.
      (document as any).getElementById('data-error').appendChild(error);
    },
    rules: {
      'defaultRTE-value': {
        maxLength:[108, 'RTE: Maximum 100 character only'],
        minLength: [15, 'RTE: Need atleast 8 character length'],
        required: [true, 'RTE: value is required']
      }
    }
  };
  formObject = new FormValidator('#myForm', option);
  (document as any).getElementById('validateSubmit').addEventListener('click', (e: any) => {
    const form = (document as any).forms.myForm;
    const formData = new FormData(form);
    const rteValue = formData.get('defaultRTE-value');
    // Use this value to the data base.
    alert(rteValue);
    e.preventDefault();
  });
});
let formObject: FormValidator;
let button: ButtonComponent;
function onChange(): void {
  button.disabled = false;
}
return (
  <form id="myForm" className="form-vertical">
    <div className="form-group">
      <RichTextEditorComponent id="defaultRTE" htmlAttributes= {{ name:
"defaultRTE-value" }} className="form-control" height={200}
showCharCount={true} maxLength={100} placeholder='Type something'
change={onChange.bind(this)} value=''>
        <Inject services={[Toolbar, Image, Link, Count, HtmlEditor,
QuickToolbar]} />
      </RichTextEditorComponent>
    </div>
    <div className="form-group">
      <div className="col-sm-3 control-label">
        <div className="col-sm-6">
          <div id='data-error' />
        </div>
      </div>
      <div className="form-btn-section">
        <ButtonComponent id="validateSubmit" ref={(scope) => { button =
scope; }} disabled={true}>Submit</ButtonComponent>
        <button id="reset-btn" className="sample-btn e-control e-btn"
type="reset" data-ripple="true">Reset</button>
      </div>
    </div>
  </form>
);
}
export default App;
{% endraw %}

```

Globalization in React Rich text editor component

Localization

The Rich Text Editor provides an option to localize its strings; it is used to adapting the editor to a particular local language. By default, the editor will use the US English (en-US) as its language. Please find the table with a list of keys and their corresponding values for the default language (en-US).

```
` javascript
'en-US': {
'richtexteditor': {
alignments: 'Alignments',
justifyLeft: 'Align Left',
justifyCenter: 'Align Center',
justifyRight: 'Align Right',
justifyFull: 'Align Justify',
fontName: 'Font Name',
fontSize: 'Font Size',
fontColor: 'Font Color',
backgroundColor: 'Background Color',
bold: 'Bold',
italic: 'Italic',
underline: 'Underline',
strikethrough: 'Strikethrough',
clearFormat: 'Clear Format',
clearAll: 'Clear All',
cut: 'Cut',
copy: 'Copy',
paste: 'Paste',
unorderedList: 'Bulleted List',
orderedList: 'Numbered List',
indent: 'Increase Indent',
outdent: 'Decrease Indent',
undo: 'Undo',
redo: 'Redo',
superscript: 'Superscript',
```

```
subscript: 'Subscript',
createLink: 'Insert Link',
openLink: 'Open Link',
editLink: 'Edit Link',
removeLink: 'Remove Link',
image: 'Insert Image',
replace: 'Replace',
align: 'Align',
caption: 'Image Caption',
remove: 'Remove',
insertLink: 'Insert Link',
display: 'Display',
altText: 'Alternative Text',
dimension: 'Change Size',
fullscreen: 'Maximize',
maximize: 'Maximize',
minimize: 'Minimize',
lowerCase: 'Lower Case',
upperCase: 'Upper Case',
print: 'Print',
formats: 'Formats',
sourcecode: 'Code View',
preview: 'Preview',
viewside: 'ViewSide',
insertCode: 'Insert Code',
linkText: 'Display Text',
linkTooltipLabel: 'Title',
linkWebUrl: 'Web Address',
linkTitle: 'Enter a title',
linkurl: 'http://example.com',
linkOpenInNewWindow: 'Open Link in New Window',
linkHeader: 'Insert Link',
dialogInsert: 'Insert',
```

```
dialogCancel: 'Cancel',
dialogUpdate: 'Update',
imageHeader: 'Insert Image',
imageLinkHeader: 'You can also provide a link from the web',
mdimageLink: 'Please provide a URL for your image',
imageUploadMessage: 'Drop image here or browse to upload',
imageDeviceUploadMessage: 'Click here to upload',
imageAlternateText: 'Alternate Text',
alternateHeader: 'Alternative Text',
browse: 'Browse',
imageUrl: 'http://example.com/image.png',
imageCaption: 'Caption',
imageSizeHeader: 'Image Size',
imageHeight: 'Height',
imageWidth: 'Width',
textPlaceholder: 'Enter Text',
inserttablebtn: 'Insert Table',
tabledialogHeader: 'Insert Table',
tableWidth: 'Width',
cellpadding: 'Cell Padding',
cellspacing: 'Cell Spacing',
columns: 'Number of columns',
rows: 'Number of rows',
tableRows: 'Table Rows',
tableColumns: 'Table Columns',
tableCellHorizontalAlign: 'Table Cell Horizontal Align',
tableCellVerticalAlign: 'Table Cell Vertical Align',
createTable: 'Create Table',
removeTable: 'Remove Table',
tableHeader: 'Table Header',
tableRemove: 'Table Remove',
tableCellBackground: 'Table Cell Background',
tableEditProperties: 'Table Edit Properties',
```

```
styles: 'Styles',
insertColumnLeft: 'Insert Column Left',
insertColumnRight: 'Insert Column Right',
deleteColumn: 'Delete Column',
insertRowBefore: 'Insert Row Before',
insertRowAfter: 'Insert Row After',
deleteRow: 'Delete Row',
tableEditHeader: 'Edit Table',
TableHeadingText: 'Heading',
TableColText: 'Col',
imageInsertLinkHeader: 'Insert Link',
editImageHeader: 'Edit Image',
alignmentsDropDownLeft: 'Align Left',
alignmentsDropDownCenter: 'Align Center',
alignmentsDropDownRight: 'Align Right',
alignmentsDropDownJustify: 'Align Justify',
imageDisplayDropDownInline: 'Inline',
imageDisplayDropDownBreak: 'Break',
tableInsertRowDropDownBefore: 'Insert row before',
tableInsertRowDropDownAfter: 'Insert row after',
tableInsertRowDropDownDelete: 'Delete row',
tableInsertColumnDropDownLeft: 'Insert column left',
tableInsertColumnDropDownRight: 'Insert column right',
tableInsertColumnDropDownDelete: 'Delete column',
tableVerticalAlignDropDownTop: 'Align Top',
tableVerticalAlignDropDownMiddle: 'Align Middle',
tableVerticalAlignDropDownBottom: 'Align Bottom',
tableStylesDropDownDashedBorder: 'Dashed Borders',
tableStylesDropDownAlternateRows: 'Alternate Rows',
pasteFormat: 'Paste Format',
pasteFormatContent: 'Choose the formatting action',
plainText: 'Plain Text',
cleanFormat: 'Clean',
```

```
keepFormat: 'Keep',
pasteDialogOk: 'OK',
pasteDialogCancel: 'Cancel',
fileManager: 'File Manager',
fileDialogHeader: 'File Browser',
formatsDropDownParagraph: 'Paragraph',
formatsDropDownCode: 'Code',
formatsDropDownQuotation: 'Quotation',
formatsDropDownHeading1: 'Heading 1',
formatsDropDownHeading2: 'Heading 2',
formatsDropDownHeading3: 'Heading 3',
formatsDropDownHeading4: 'Heading 4',
fontNameSegoeUI: 'Segoe UI',
fontNameArial: 'Arial',
fontNameGeorgia: 'Georgia',
fontNameImpact: 'Impact',
fontNameTahoma: 'Tahoma',
fontNameTimesNewRoman: 'Times New Roman',
fontNameVerdana: 'Verdana',
numberFormatListNumber: 'Number',
numberFormatListLowerAlpha: 'LowerAlpha',
numberFormatListUpperAlpha: 'UpperAlpha',
numberFormatListLowerRoman: 'LowerRoman',
numberFormatListUpperRoman: 'UpperRoman',
numberFormatListLowerGreek: 'LowerGreek',
bulletFormatListDisc: 'Disc',
bulletFormatListCircle: 'Circle',
bulletFormatListSquare: 'Square',
numberFormatListNone: 'None',
bulletFormatListNone: 'None',
formatPainter: 'Format Painter',
emojiPicker: 'Emoji Picker',
embeddedCode: 'Embedded Code',
```

```

pasteEmbeddedCodeHere: 'Paste Embedded Code here',
emojiPickerTypeToFind: 'Type to find',
emojiPickerNoResultFound: 'No results found',
emojiPickerTrySomethingElse: 'Try something else',
}
}
,

```

To localize the editor's strings with your own localization, copy the default language informations and localize the strings in the values column. For example, to localize the editor in German language ("de-DE").

```

` javascript
'de-DE': {
  'richtexteditor': {
    alignments: 'Alignments',
    justifyLeft: 'Ausrichten von Text links',
    justifyCenter: 'Text-Zentrum',
    justifyRight: 'Ausrichten von Text rechts',
    justifyFull: 'rechtfertigen',
    fontName: 'Wählen Sie Schriftfamilie',
    fontSize: 'Wählen Sie Schriftgröße',
    fontColor: 'Wählen Sie die Farbe',
    backgroundColor: 'Hintergrundfarbe',
    bold: 'fett',
    italic: 'kursiv',
    underline: 'unterstreichen',
    strikethrough: 'Durchgestrichen',
    clearAll: 'Alles',
    clearFormat: 'Klar Format',
    cut: 'schneiden',
    copy: 'Kopieren',
    paste: 'Paste',
    unorderedList: 'Legen Sie ungeordnete Liste',
    orderedList: 'Geordnete Liste einfügen',
    indent: 'Einzug',

```

outdent: 'Einzug verkleinern',
undo: 'lösen',
redo: 'Wiederherstellen',
superscript: 'Überschrift',
subscript: 'index',
createLink: 'Link einfügen',
removeLink: 'fjern Hyperlink',
openLink: 'Open link',
editLink: 'Edit link',
image: 'Bild einfügen',
replace: 'ersetzen',
align: 'ausrichten',
caption: 'Bildbeschriftung',
formats: 'Formats',
remove: 'Löschen',
insertLink: 'Link einfügen',
display: 'Anzeige',
alttext: 'alternativer Text',
dimension: 'Größe',
fullscreen: 'Vollbild',
maximize: 'Maximieren',
minimize: 'minimieren',
zoomIn: 'hineinzoomen',
zoomOut: 'Rauszoomen',
upperCase: 'Großbuchstaben',
lowerCase: 'Kleinbuchstaben',
print: 'Drucken',
sourcecode: 'Quellcode',
preview: 'Vorschau',
viewside: 'Seite anzeigen',
insertcode: 'Code eingeben',
linkText: 'Displaytekst',
linkTooltipLabel: 'tooltip',

linkWebUrl: 'Webadres',
linkOpenInNewWindow: 'Open de link in een nieuw venster',
linkHeader: 'Link invoegen',
dialogInsert: 'invoegen',
dialogCancel: 'Annuleer',
dialogUpdate: 'Bijwerken',
imageHeader: 'Voeg afbeelding in',
imageLinkHeader: 'U kunt ook een link van internet opgeven',
imageUploadMessage: 'Zet hier een afbeelding neer of klik om te uploaden',
imageDeviceUploadMessage: 'Klik hier om te uploaden',
imageAlternateText: 'Alternatieve tekst',
alternateHeader: 'Alternatieve tekst',
browse: 'Blader',
imageUrl: 'URL',
imageCaption: 'onderschrift',
imageSizeHeader: 'Afbeeldingsgrootte',
imageHeight: 'Hoogte',
imageWidth: 'Breedte',
textPlaceholder: 'Text eingeben',
inserttablebtn: 'Tabelle einfügen',
tabledialogHeader: 'Tabelle einfügen',
tableWidth: 'Breite',
cellpadding: 'Zellauffüllung',
cellspacing: 'Zellabstand',
columns: 'Anzahl der Spalten',
rows: 'Reihenanzahl',
tableRows: 'Tabellenzeilen',
tableColumns: 'Tabellenspalten',
tableCellHorizontalAlign: 'Horizontale Ausrichtung der Tabellenzelle',
tableCellVerticalAlign: 'Vertikale Ausrichtung der Tabellenzelle',
createTable: 'Tabelle erstellen',
removeTable: 'Tabelle entfernen',
tableHeader: 'Tabellenkopfzeile',

tableRemove: 'Tabelle entfernen',
tableCellBackground: 'Tabellenzellenhintergrund',
tableEditProperties: 'Eigenschaften der Tabellenbearbeitung',
styles: 'Styles',
insertColumnLeft: 'Spalte links einfügen',
insertColumnRight: 'Spalte rechts einfügen',
deleteColumn: 'Spalte löschen',
insertRowBefore: 'Zeile vor einfügen',
insertRowAfter: 'Zeile einfügen nach',
deleteRow: 'Zeile löschen',
tableEditHeader: 'Tabelle bearbeiten',
TableHeadingText: 'Überschrift',
TableColText: 'Col',
imageInsertLinkHeader: 'Link einfügen',
editImageHeader: 'Bild bearbeiten',
alignmentsDropDownLeft: 'Linksbündig',
alignmentsDropDownCenter: 'Im Zentrum anordnen',
alignmentsDropDownRight: 'Rechts ausrichten',
alignmentsDropDownJustify: 'Justize ausrichten',
imageDisplayDropDownInline: 'In der Reihe',
imageDisplayDropDownBreak: 'Brechen',
tableInsertRowDropDownBefore: 'Reihe vorher einfügen',
tableInsertRowDropDownAfter: 'Zeile danach einfügen',
tableInsertRowDropDownDelete: 'Zeile löschen',
tableInsertColumnDropDownLeft: 'Spalte links einfügen',
tableInsertColumnDropDownRight: 'Spalte rechts einfügen',
tableInsertColumnDropDownDelete: 'Spalte löschen',
tableVerticalAlignDropDownTop: 'Top ausrichten',
tableVerticalAlignDropDownMiddle: 'Mitte ausrichten',
tableVerticalAlignDropDownBottom: 'Unten ausrichten',
tableStylesDropDownDashedBorder: 'Gestrichelte Grenzen',
tableStylesDropDownAlternateRows: 'Alternative Zeilen',
pasteFormat: 'Format einfügen',

```
pasteFormatContent: 'Wählen Sie die Formatierungsaktion aus',
plainText: 'Einfacher Text',
cleanFormat: 'sauber',
keepFormat: 'Behalten',
formatsDropDownParagraph: 'Absatz',
formatsDropDownCode: 'Kodex',
formatsDropDownQuotation: 'Zitat',
formatsDropDownHeading1: 'Überschrift 1',
formatsDropDownHeading2: 'Überschrift 2',
formatsDropDownHeading3: 'Überschrift 3',
formatsDropDownHeading4: 'Überschrift 4',
fontNameSegoeUI: 'Segoe UI',
fontNameArial: 'Arial',
fontNameGeorgia: 'Georgia',
fontNameImpact: 'Einschlag',
fontNameTahoma: 'Tahoma',
fontNameTimesNewRoman: 'Mal Neu römisch',
fontNameVerdana: 'Verdana'
numberFormatListNumber: 'Nummer',
numberFormatListLowerAlpha: 'Unteres Alpha',
numberFormatListUpperAlpha: 'Oberes Alpha',
numberFormatListLowerRoman: 'Niederrömisch',
numberFormatListUpperRoman: 'Oberrömisch',
numberFormatListLowerGreek: 'Niedergriechisch',
bulletFormatListDisc: 'Rabatt',
bulletFormatListCircle: 'Kreis',
bulletFormatListSquare: 'Platz',
numberFormatListNone: 'Keiner',
bulletFormatListNone: 'Keiner'
}
},
,
```

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - Localization Sample
 */
import { L10n } from '@syncfusion/ej2-base';
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
L10n.load({
  'de-DE': {
    'richtexteditor': {
      align: "ausrichten",
      alignments: "Alignments",
      alternateHeader: "Alternatieve tekst",
      alttext: "alternativer Text",
      backgroundColor: "Hintergrundfarbe",
      bold: "fett",
      browse: "Blader",
      caption: "Bildbeschriftung",
      clearAll: "Alles",
      clearFormat: "Klar Format",
      copy: "Kopieren",
      createLink: "Link einfügen",
      cut: "schneiden",
      dialogCancel: "Annuleer",
      dialogInsert: "invoegen",
      dialogUpdate: "Bijwerken",
      dimension: "Größe",
      display: "Anzeige",
      editLink: "Edit link",
      fontColor: "Wählen Sie die Farbe",
      fontName: "Wählen Sie Schriftfamilie",
      fontSize: "Wählen Sie Schriftgröße",
      formats: "Formats",
      fullscreen: "Vollbild",
      image: "Bild einfügen",
      imageAlternateText: "Alternatieve tekst",
      imageCaption: "onderschrift",
      imageDeviceUploadMessage: "Klik hier om te uploaden",
      imageHeader: "Voeg afbeelding in",
      imageHeight: "Hoogte",
      imageLinkHeader: "U kunt ook een link van internet opgeven",
      imageSizeHeader: "Afbeeldingsgrootte",
      imageUploadMessage: "Zet hier een afbeelding neer of klik om te
uploaden",
      imageUrl: "URL",
      imageWidth: "Breedte",
      indent: "Einzug",
      insertLink: "Link einfügen",
      insertcode: "Code eingeben",
      italic: "kursiv",
      justifyCenter: "Text-Zentrum",
      justifyFull: "rechtfertigen",
      justifyLeft: "Ausrichten von Text links",
      justifyRight: "Ausrichten von Text rechts",

```

```

        linkHeader: "Link invoegen",
        linkOpenInNewWindow: "Open de link in een nieuw venster",
        linkText: "Displaytekst",
        linkTooltipLabel: "tooltip",
        linkWebUrl: "Webadres",
        lowerCase: "Kleinbuchstaben",
        maximize: "Maximieren",
        minimize: "minimieren",
        openLink: "Open link",
        orderedList: "Geordnete Liste einfügen",
        outdent: "Einzug verkleinern",
        paste: "Paste",
        preview: "Vorschau",
        print: "Drucken",
        redo: "Wiederherstellen",
        remove: "Löschen",
        removeLink: "fjern Hyperlink",
        replace: "ersetzen",
        sourcecode: "Quellcode",
        strikethrough: "Durchgestrichen",
        subscript: "index",
        superscript: "Überschrift",
        underline: "unterstreichen",
        undo: "lösen",
        unorderedList: "Legen Sie ungeordnete Liste",
        upperCase: "Großbuchstaben",
        viewside: "Seite anzeigen",
        zoomIn: "hineinzoomen",
        zoomOut: "Rauszoomen",
        formatsDropDownParagraph: "Absatz",
        formatsDropDownCode: "Kodex",
        formatsDropDownQuotation: "Zitat",
        formatsDropDownHeading1: "Überschrift 1",
        formatsDropDownHeading2: "Überschrift 2",
        formatsDropDownHeading3: "Überschrift 3",
        formatsDropDownHeading4: "Überschrift 4",
        fontNameSegoeUI: "Segoe UI",
        fontNameArial: "Arial",
        fontNameGeorgia: "Georgia",
        fontNameImpact: "Einschlag",
        fontNameTahoma: "Tahoma",
        fontNameTimesNewRoman: "Mal Neu römisch",
        fontNameVerdana: "Verdana"
    }
  }
});
class App extends React.Component {
  render() {
    return (
      <RichTextEditorComponent height={450} locale={'de-DE'}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides the best user experience to create and update the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>

```

```

        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
    </RichTextEditorComponent>;
  }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Localization Sample
 */
import { L10n } from '@syncfusion/ej2-base';
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
L10n.load({
  'de-DE': {
    'richtexteditor': {
      align: "ausrichten",
      alignments: "Alignments",
      alternateHeader: "Alternatieve tekst",
      alttext: "alternativer Text",
      backgroundColor: "Hintergrundfarbe",

```

```

bold: "fett",
browse: "Blader",
caption: "Bildbeschriftung",
clearAll: "Alles",
clearFormat: "Klar Format",
copy: "Kopieren",
createLink: "Link einfügen",
cut: "schneiden",
dialogCancel: "Annuleer",
dialogInsert: "invoegen",
dialogUpdate: "Bijwerken",
dimension: "Größe",
display: "Anzeige",
editLink: "Edit link",
fontColor: "Wählen Sie die Farbe",
fontName: "Wählen Sie Schriftfamilie",
fontSize: "Wählen Sie Schriftgröße",
formats: "Formats",
fullscreen: "Vollbild",
image: "Bild einfügen",
imageAlternateText: "Alternatieve tekst",
imageCaption: "onderschrift",
imageDeviceUploadMessage: "Klik hier om te uploaden",
imageHeader: "Voeg afbeelding in",
imageHeight: "Hoogte",
imageLinkHeader: "U kunt ook een link van internet opgeven",
imageSizeHeader: "Afbeeldingsgrootte",
imageUploadMessage: "Zet hier een afbeelding neer of klik om te
uploaden",
imageUrl: "URL",
imageWidth: "Breedte",
indent: "Einzug",
insertLink: "Link einfügen",
insertcode: "Code eingeben",
italic: "kursiv",
justifyCenter: "Text-Zentrum",
justifyFull: "rechtfertigen",
justifyLeft: "Ausrichten von Text links",
justifyRight: "Ausrichten von Text rechts",
linkHeader: "Link invoegen",
linkOpenInNewWindow: "Open de link in een nieuw venster",
linkText: "Displaytekst",
linkTooltipLabel: "tooltip",
linkWebUrl: "Webadres",
lowerCase: "Kleinbuchstaben",
maximize: "Maximieren",
minimize: "minimieren",
openLink: "Open link",
orderedList: "Geordnete Liste einfügen",
outdent: "Einzug verkleinern",
paste: "Paste",
preview: "Vorschau",
print: "Drucken",
redo: "Wiederherstellen",
remove: "Löschen",
removeLink: "fjern Hyperlink",
replace: "ersetzen",

```

```

        sourcecode: "Quellcode",
        strikethrough: "Durchgestrichen",
        subscript: "index",
        superscript: "Überschrift",
        underline: "unterstreichen",
        undo: "lösen",
        unorderedList: "Legen Sie ungeordnete Liste",
        upperCase: "Großbuchstaben",
        viewside: "Seite anzeigen",
        zoomIn: "hineinzoomen",
        zoomOut: "Rauszoomen",
        formatsDropDownParagraph: "Absatz",
        formatsDropDownCode: "Kodex",
        formatsDropDownQuotation: "Zitat",
        formatsDropDownHeading1: "Überschrift 1",
        formatsDropDownHeading2: "Überschrift 2",
        formatsDropDownHeading3: "Überschrift 3",
        formatsDropDownHeading4: "Überschrift 4",
        fontNameSegoeUI: "Segoe UI",
        fontNameArial: "Arial",
        fontNameGeorgia: "Georgia",
        fontNameImpact: "Einschlag",
        fontNameTahoma: "Tahoma",
        fontNameTimesNewRoman: "Mal Neu römisch",
        fontNameVerdana: "Verdana"
    }
}
});
class App extends React.Component<{},{}> {
    public render() {
        return (
            <RichTextEditorComponent height={450} locale={'de-DE'}>
                <p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides the best user experience to create and update the content.
                Users can format their content using standard toolbar commands.</p>
                <p><b>Key features:</b></p>
                <ul>
                    <li>
                        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
                    </li>
                    <li>
                        <p>Capable of handling markdown editing.</p>
                    </li>
                    <li>
                        <p>Contains a modular library to load the necessary functionality on demand.</p>
                    </li>
                    <li>
                        <p>Provides a fully customizable toolbar.</p>
                    </li>
                    <li>
                        <p>Provides HTML view to edit the source directly for developers.</p>
                    </li>
                    <li>
                        <p>Supports third-party library integration.</p>
                    </li>
                </ul>
            </RichTextEditorComponent>
        );
    }
}

```



```

        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Localization Sample
 */
import { L10n } from '@syncfusion/ej2-base';
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
L10n.load({
  'de-DE': {
    'richtexteditor': {
      align: "ausrichten",
      alignments: "Alignments",
      alternateHeader: "Alternatieve tekst",
      alttext: "alternativer Text",
      backgroundColor: "Hintergrundfarbe",
      bold: "fett",
      browse: "Blader",
      caption: "Bildbeschriftung",
      clearAll: "Alles",
      clearFormat: "Klar Format",
      copy: "Kopieren",
      createLink: "Link einfügen",
      cut: "schneiden",
      dialogCancel: "Annuleer",
      dialogInsert: "invoegen",
      dialogUpdate: "Bijwerken",
      dimension: "Größe",
      display: "Anzeige",

```

```

editLink: "Edit link",
fontColor: "Wählen Sie die Farbe",
fontName: "Wählen Sie Schriftfamilie",
fontSize: "Wählen Sie Schriftgröße",
formats: "Formats",
fullscreen: "Vollbild",
image: "Bild einfügen",
imageAlternateText: "Alternatieve tekst",
imageCaption: "onderschrift",
imageDeviceUploadMessage: "Klik hier om te uploaden",
imageHeader: "Voeg afbeelding in",
imageHeight: "Hoogte",
imageLinkHeader: "U kunt ook een link van internet opgeven",
imageSizeHeader: "Afbeeldingsgrootte",
imageUploadMessage: "Zet hier een afbeelding neer of klik om te
uploaden",
imageUrl: "URL",
imageWidth: "Breedte",
indent: "Einzug",
insertLink: "Link einfügen",
insertcode: "Code eingeben",
italic: "kursiv",
justifyCenter: "Text-Zentrum",
justifyFull: "rechtfertigen",
justifyLeft: "Ausrichten von Text links",
justifyRight: "Ausrichten von Text rechts",
linkHeader: "Link invoegen",
linkOpenInNewWindow: "Open de link in een nieuw venster",
linkText: "Displaytekst",
linkTooltipLabel: "tooltip",
linkWebUrl: "Webadres",
lowerCase: "Kleinbuchstaben",
maximize: "Maximieren",
minimize: "minimieren",
openLink: "Open link",
orderedList: "Geordnete Liste einfügen",
outdent: "Einzug verkleinern",
paste: "Paste",
preview: "Vorschau",
print: "Drucken",
redo: "Wiederherstellen",
remove: "Löschen",
removeLink: "fjern Hyperlink",
replace: "ersetzen",
sourcecode: "Quellcode",
strikethrough: "Durchgestrichen",
subscript: "index",
superscript: "Überschrift",
underline: "unterstreichen",
undo: "lösen",
unorderedList: "Legen Sie ungeordnete Liste",
upperCase: "Großbuchstaben",
viewside: "Seite anzeigen",
zoomIn: "hineinzoomen",
zoomOut: "Rauszoomen",
formatsDropDownParagraph: "Absatz",
formatsDropDownCode: "Kodex",

```

```

        formatsDropDownQuotation: "Zitat",
        formatsDropDownHeading1: "Überschrift 1",
        formatsDropDownHeading2: "Überschrift 2",
        formatsDropDownHeading3: "Überschrift 3",
        formatsDropDownHeading4: "Überschrift 4",
        fontNameSegoeUI: "Segoe UI",
        fontNameArial: "Arial",
        fontNameGeorgia: "Georgia",
        fontNameImpact: "Einschlag",
        fontNameTahoma: "Tahoma",
        fontNameTimesNewRoman: "Mal Neu römisch",
        fontNameVerdana: "Verdana"
    }
}
});
function App() {
    return (<RichTextEditorComponent height={450} locale={'de-DE'}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
                <p>Capable of handling markdown editing.</p>
            </li>
            <li>
                <p>Contains a modular library to load the necessary functionality
on demand.</p>
            </li>
            <li>
                <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
                <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
                <p>Supports third-party library integration.</p>
            </li>
            <li>
                <p>Allows preview of modified content before saving it.</p>
            </li>
            <li>
                <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
            </li>
            <li>
                <p>Contains undo/redo manager.</p>
            </li>
            <li>
                <p>Creates bulleted and numbered lists.</p>
            </li>
        </ul>
    </RichTextEditorComponent>);
}

```

```

        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
        </RichTextEditorComponent>);
    }
    export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Localization Sample
 */
import { L10n } from '@syncfusion/ej2-base';
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
L10n.load({
    'de-DE': {
        'richtexteditor': {
            align: "ausrichten",
            alignments: "Alignments",
            alternateHeader: "Alternatieve tekst",
            alttext: "alternativer Text",
            backgroundColor: "Hintergrundfarbe",
            bold: "fett",
            browse: "Blader",
            caption: "Bildbeschriftung",
            clearAll: "Alles",
            clearFormat: "Klar Format",
            copy: "Kopieren",
            createLink: "Link einfügen",
            cut: "schneiden",
            dialogCancel: "Annuleer",
            dialogInsert: "invoegen",
            dialogUpdate: "Bijwerken",
            dimension: "Größe",
            display: "Anzeige",
            editLink: "Edit link",
            fontColor: "Wählen Sie die Farbe",
            fontName: "Wählen Sie Schriftfamilie",
            fontSize: "Wählen Sie Schriftgröße",
            formats: "Formats",
            fullscreen: "Vollbild",
            image: "Bild einfügen",
            imageAlternateText: "Alternatieve tekst",
            imageCaption: "onderschrift",
            imageDeviceUploadMessage: "Klik hier om te uploaden",
            imageHeader: "Voeg afbeelding in",
            imageHeight: "Hoogte",
            imageLinkHeader: "U kunt ook een link van internet opgeven",
            imageSizeHeader: "Afbeeldingsgrootte",
            imageUploadMessage: "Zet hier een afbeelding neer of klik om te uploaden",
            imageUrl: "URL",
            imageWidth: "Breedte",
            indent: "Einzug",
            insertLink: "Link einfügen",

```

```

insertcode: "Code eingeben",
italic: "kursiv",
justifyCenter: "Text-Zentrum",
justifyFull: "rechtfertigen",
justifyLeft: "Ausrichten von Text links",
justifyRight: "Ausrichten von Text rechts",
linkHeader: "Link invoegen",
linkOpenInNewWindow: "Open de link in een nieuw venster",
linkText: "Displaytekst",
linkTooltipLabel: "tooltip",
linkWebUrl: "Webadres",
lowerCase: "Kleinbuchstaben",
maximize: "Maximieren",
minimize: "minimieren",
openLink: "Open link",
orderedList: "Geordnete Liste einfügen",
outdent: "Einzug verkleinern",
paste: "Paste",
preview: "Vorschau",
print: "Drucken",
redo: "Wiederherstellen",
remove: "Löschen",
removeLink: "fjern Hyperlink",
replace: "ersetzen",
sourcecode: "Quellcode",
strikethrough: "Durchgestrichen",
subscript: "index",
superscript: "Überschrift",
underline: "unterstreichen",
undo: "lösen",
unorderedList: "Legen Sie ungeordnete Liste",
upperCase: "Großbuchstaben",
viewside: "Seite anzeigen",
zoomIn: "hineinzoomen",
zoomOut: "Rauszoomen",
formatsDropDownParagraph: "Absatz",
formatsDropDownCode: "Kodex",
formatsDropDownQuotation: "Zitat",
formatsDropDownHeading1: "Überschrift 1",
formatsDropDownHeading2: "Überschrift 2",
formatsDropDownHeading3: "Überschrift 3",
formatsDropDownHeading4: "Überschrift 4",
fontNameSegoeUI: "Segoe UI",
fontNameArial: "Arial",
fontNameGeorgia: "Georgia",
fontNameImpact: "Einschlag",
fontNameTahoma: "Tahoma",
fontNameTimesNewRoman: "Mal Neu römisch",
fontNameVerdana: "Verdana"
}
}
});
function App() {
  return (
    <RichTextEditorComponent height={450} locale={'de-DE'}>

```

```

    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
    you get") editor that provides the best user experience to create and update
    the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
  />
</RichTextEditorComponent>
);
}
export default App;

```

RTL

Specifies the direction of the Rich Text Editor component using the [enableRtl](#) property. For writing systems that require it like Arabic, Hebrew, etc., the direction can be switched to right-to-left.

Note: It will not change based on the locale property.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - RTL Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  render() {
    return (<RichTextEditorComponent height={450} enableRtl={true}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
    </RichTextEditorComponent>);
  }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - RTL Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}> {
  public render() {
    return (
      <RichTextEditorComponent height={450} enableRtl={true}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
          </li>
          <li>
            <p>Contains undo/redo manager.</p>
          </li>
          <li>
            <p>Creates bulleted and numbered lists.</p>
          </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
      />
    </RichTextEditorComponent>
  );
}

```



```

    }
  }
  export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - RTL Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  return (<RichTextEditorComponent height={450} enableRtl={true}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
  </RichTextEditorComponent>

```

```

        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
    </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - RTL Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    return (
        <RichTextEditorComponent height={450} enableRtl={true}>
            <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
                Users can format their content using standard toolbar commands.</p>
                <p><b>Key features:</b></p>
                <ul>
                    <li>
                        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
                    </li>
                    <li>
                        <p>Capable of handling markdown editing.</p>
                    </li>
                    <li>
                        <p>Contains a modular library to load the necessary functionality
on demand.</p>
                    </li>
                    <li>
                        <p>Provides a fully customizable toolbar.</p>
                    </li>
                    <li>
                        <p>Provides HTML view to edit the source directly for
developers.</p>
                    </li>
                    <li>
                        <p>Supports third-party library integration.</p>
                    </li>
                    <li>
                        <p>Allows preview of modified content before saving it.</p>
                    </li>
                    <li>
                        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
                    </li>
                    <li>
                        <p>Contains undo/redo manager.</p>
                    </li>
                    <li>
                        <p>Creates bulleted and numbered lists.</p>
                    </li>
                </ul>
            </RichTextEditorComponent>
        </App>
    );
}

```

```

        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
    />
  </RichTextEditorComponent>
);
}
export default App;

```

Miscellaneous in React Rich text editor component

Placeholder

Specifies the placeholder for the Rich Text Editor's content used when the Rich Text Editor body is empty through the [placeholder](#) property.

Through the `e-rte-placeholder` class to define our custom font family, font color, and styles to the placeholder text.

```

`css

.e-richtexteditor .e-rte-placeholder {
font-family: monospace;
}
`

```

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - Placeholder Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  render() {
    return (
      <RichTextEditorComponent placeholder={'Type Something'}>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
      </RichTextEditorComponent>
    );
  }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Placeholder Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditor, RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';

```

```

class App extends React.Component<{}, {}> {
  public render() {
    return (
      <RichTextEditorComponent placeholder={'Type Something'}>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
      />
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Placeholder Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  return (<RichTextEditorComponent placeholder={'Type Something'}>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
  </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Placeholder Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditor, RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-
react-richtexteditor';
import * as React from 'react';
function App() {
  return (
    <RichTextEditorComponent placeholder={'Type Something'}>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
    />
    </RichTextEditorComponent>
  );
}
export default App;

```

Character count

The Rich Text Editor automatically counts the number of characters in the content are while typing using the [showCharCount](#) property. The characters count displayed at the bottom of the editor. You can limit the number of characters in your content using the [maxLength](#) property. By default, the editor sets the characters limit value is infinity.

The character count color will be modified based on the characters in the Rich Text Editor.

| **Status** | **Description** |

| --- | --- |

| Normal | Till 70% of given maxLength value, character count color is black. |

| Warning | Once the number of character count in the Rich Text Editor reached 70% of given maxLength value, the character count color will be orange, indicating that, the count value going to reach the maximum count. |

| Error | Once the number of character count in the Rich Text Editor reached 90% of given maxLength value, the character count color will be red, indicating that, the count value reached the maximum count. |

To create Rich Text Editor with character count feature, inject the Count module to the RTE using the `RichTextEditor.Inject(Count)` method.

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - Character Count Sample
 */
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  value() {
    return (
      <div>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
          Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
        </ul>
      </div>
    );
  }
}
```

```

        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
    </div>);
  }
;
  render() {
    return (<RichTextEditorComponent height={450} showCharCount={true}
maxLength={2000} valueTemplate={this.value}>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]}/>
      </RichTextEditorComponent>);
  }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Character Count Sample
 */
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  public value(): JSX.Element {
    return(<div>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
      </ul>
    </div>);
  }
}

```

```

        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
    </div>);
  };
  public render() {
    return (
      <RichTextEditorComponent height={450} showCharCount={true}
maxLength={2000} valueTemplate = {this.value}>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]} />
      </RichTextEditorComponent>
    );
  }
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Character Count Sample
 */
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  function value() {
    return <div>
      <p>The Rich Text Editor component is WYSIWYG ("what you red is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
    </div>
  }
}

```

```

    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
  </div>);
}
;
  return <RichTextEditorComponent height={450} showCharCount={true}
maxLength={2000} valueTemplate={value}>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]}/>
    </RichTextEditorComponent>;
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Character Count Sample
 */
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';

```



```

function App() {
  function value(): JSX.Element {
    return(<div>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
    </div>);
  };
  return (
    <RichTextEditorComponent height={450} showCharCount={true}
maxLength={2000} valueTemplate = {value}>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]} />
    </RichTextEditorComponent>
  );
}
export default App;

```

Code view

Rich Text Editor includes the ability for users to directly edit HTML code via **Source View** in the text area. If you made any modification in Source view directly, the changes will be reflected in the Rich Text Editor's content. So, the users will have more flexibility over the content they have created.

[Class-component]**APP.JSX**

```
{% raw %}
/**
 * Rich Text Editor - CodeView sample
 */
import { createElement } from '@syncfusion/ej2-base';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import CodeMirror from 'codemirror';
import 'codemirror/mode/css/css.js';
import 'codemirror/mode/htmlmixed/htmlmixed.js';
import 'codemirror/mode/javascript/javascript';
import * as React from 'react';
class App extends React.Component {
  myCodeMirror;
  textArea;
  rteObj;
  toolbarSettings = {
    items: ['SourceCode']
  };
  value() {
    return (<div>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.

      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
      </ul>
    </div>
  )
}
```

```

        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
    </div>);
  }
;
  mirrorConversion(e) {
    const id = this.rteObj.getID() + 'mirror-view';
    let mirrorView = this.rteObj.element.querySelector('#' + id);
    const charCount = this.rteObj.element.querySelector('.e-rte-
character-count');
    if (e.targetItem === 'Preview') {
      this.textArea.style.display = 'block';
      mirrorView.style.display = 'none';
      this.textArea.innerHTML = this.myCodeMirror.getValue();
      charCount.style.display = 'block';
    }
    else {
      if (!mirrorView) {
        mirrorView = createElement('div', { className: 'e-content'
});
        mirrorView.id = id;
        this.textArea.parentNode.appendChild(mirrorView);
      }
      else {
        mirrorView.innerHTML = '';
      }
      this.textArea.style.display = 'none';
      mirrorView.style.display = 'block';
      this.renderCodeMirror(mirrorView, this.rteObj.value);
      charCount.style.display = 'none';
    }
  }
  renderCodeMirror(mirrorView, content) {
    this.myCodeMirror = CodeMirror(mirrorView, {
      lineNumbers: true,
      lineWrapping: true,
      mode: 'text/html',
      value: content
    });
  }
  actionComplete(e) {
    this.textArea = this.rteObj.contentModule.getEditPanel();
    if (e.targetItem && (e.targetItem === 'SourceCode' || e.targetItem
=== 'Preview')) {
      this.rteObj.sourceCodeModule.getPanel().style.display = 'none';

```

```

        this.mirrorConversion(e);
    }
    else {
        setTimeout(() => {
            this.rteObj.toolbarModule.refreshToolbarOverflow();
        }, 400);
    }
}
created() {
    this.textArea = this.rteObj.contentModule.getEditPanel();
}
render() {
    return (<RichTextEditorComponent ref={(richtexteditor) => {
this.rteObj = richtexteditor; }} height={450}
toolbarSettings={this.toolbarSettings} showCharCount={true}
actionComplete={this.actionComplete = this.actionComplete.bind(this)}
created={this.created = this.created.bind(this)}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
                <p>Capable of handling markdown editing.</p>
            </li>
            <li>
                <p>Contains a modular library to load the necessary functionality
on demand.</p>
            </li>
            <li>
                <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
                <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
                <p>Supports third-party library integration.</p>
            </li>
            <li>
                <p>Allows preview of modified content before saving it.</p>
            </li>
            <li>
                <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
            </li>
            <li>
                <p>Contains undo/redo manager.</p>
            </li>
            <li>
                <p>Creates bulleted and numbered lists.</p>
            </li>
        </ul>
    </RichTextEditorComponent>
    );
}

```

```

        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]}/>
        </RichTextEditorComponent>);
    }
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - CodeView sample
 */
import { createElement } from '@syncfusion/ej2-base';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import CodeMirror from 'codemirror';
import 'codemirror/mode/css/css.js';
import 'codemirror/mode/htmlmixed/htmlmixed.js';
import 'codemirror/mode/javascript/javascript';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    public myCodeMirror: any;
    public textArea: HTMLInputElement;
    public rteObj: RichTextEditorComponent;
    public toolbarSettings: object = {
        items: ['SourceCode']
    }
    public value(): JSX.Element {
        return (<div>
            <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.

            Users can format their content using standard toolbar commands.</p>
            <p><b>Key features:</b></p>
            <ul>
                <li>
                    <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
                </li>
                <li>
                    <p>Capable of handling markdown editing.</p>
                </li>
                <li>
                    <p>Contains a modular library to load the necessary functionality
on demand.</p>
                </li>
                <li>
                    <p>Provides a fully customizable toolbar.</p>
                </li>
                <li>
                    <p>Provides HTML view to edit the source directly for
developers.</p>
                </li>
                <li>

```

```

        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
  </div>);
};
public mirrorConversion(e: any): void {
  const id: string = this.rteObj.getID() + 'mirror-view';
  let mirrorView: HTMLElement = this.rteObj.element.querySelector('#' + id)
as HTMLElement;
  const charCount: HTMLElement = this.rteObj.element.querySelector('.e-rte-
character-count') as HTMLElement;
  if (e.targetItem === 'Preview') {
    this.textArea.style.display = 'block';
    mirrorView.style.display = 'none';
    this.textArea.innerHTML = this.myCodeMirror.getValue();
    charCount.style.display = 'block';
  } else {
    if (!mirrorView) {
      mirrorView = createElement('div', { className: 'e-content' });
      mirrorView.id = id;
      (this.textArea as any).parentNode.appendChild(mirrorView);
    } else {
      mirrorView.innerHTML = '';
    }
    this.textArea.style.display = 'none';
    mirrorView.style.display = 'block';
    this.renderCodeMirror(mirrorView, this.rteObj.value);
    charCount.style.display = 'none';
  }
}
public renderCodeMirror(mirrorView: HTMLElement, content: string): void {
  this.myCodeMirror = CodeMirror(mirrorView, {
    lineNumbers: true,
    lineWrapping: true,
    mode: 'text/html',
    value: content
  });
}
public actionComplete(e: any) {
  this.textArea = (this.rteObj as any).contentModule.getEditPanel() as
HTMLElement;
  if (e.targetItem && (e.targetItem === 'SourceCode' || e.targetItem ===
'Preview')) {

```

```

    (this.rteObj.sourceCodeModule.getPanel() as any).style.display =
    'none';
    this.mirrorConversion(e);
  } else {
    setTimeout(() => {
      this.rteObj.toolbarModule.refreshToolbarOverflow();
    }, 400);
  }
}
}
public created(): void {
  this.textArea = (this.rteObj as any).contentModule.getEditPanel() as
  HTMLInputElement;
}
public render() {
  return (
    <RichTextEditorComponent ref={ (richtexteditor) => { this.rteObj =
    richtexteditor! } } height={450} toolbarSettings={this.toolbarSettings}
    showCharCount={true} actionComplete={this.actionComplete
    =this.actionComplete.bind(this) } created={this.created =
    this.created.bind(this)}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
      you get") editor that provides the best user experience to create and update
      the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
          on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
          developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
          etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>

```

```

        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]} />
  </RichTextEditorComponent>
);
}
}
export default App;
{% endraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - CodeView sample
 */
import { createElement } from '@syncfusion/ej2-base';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import CodeMirror from 'codemirror';
import 'codemirror/mode/css/css.js';
import 'codemirror/mode/htmlmixed/htmlmixed.js';
import 'codemirror/mode/javascript/javascript';
import * as React from 'react';
import { useEffect } from "react";
function App() {
  useEffect(() => {
    return () => {
      <div>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary
functionality on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>

```



```

        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
    </div>;
  };
});
let myCodeMirror;
let textArea;
let rteObj;
let toolbarSettings = {
  items: ['SourceCode']
};
function value() {
  return ();
}
;
function mirrorConversion(e) {
  const id = rteObj.getID() + 'mirror-view';
  let mirrorView = rteObj.element.querySelector('#' + id);
  const charCount = rteObj.element.querySelector('.e-rte-character-
count');
  if (e.targetItem === 'Preview') {
    textArea.style.display = 'block';
    mirrorView.style.display = 'none';
    textArea.innerHTML = myCodeMirror.getValue();
    charCount.style.display = 'block';
  }
  else {
    if (!mirrorView) {
      mirrorView = createElement('div', { className: 'e-content'
});
      mirrorView.id = id;
      textArea.parentNode.appendChild(mirrorView);
    }
    else {
      mirrorView.innerHTML = '';
    }
    textArea.style.display = 'none';
    mirrorView.style.display = 'block';
    renderCodeMirror(mirrorView, rteObj.value);
    charCount.style.display = 'none';
  }
}

```

```

    }
    function renderCodeMirror(mirrorView, content) {
      myCodeMirror = CodeMirror(mirrorView, {
        lineNumbers: true,
        lineWrapping: true,
        mode: 'text/html',
        value: content
      });
    }
    function actionComplete(e) {
      textArea = rteObj.contentModule.getEditPanel();
      if (e.targetItem && (e.targetItem === 'SourceCode' || e.targetItem === 'Preview')) {
        rteObj.sourceCodeModule.getPanel().style.display = 'none';
        mirrorConversion(e);
      }
      else {
        setTimeout(() => {
          rteObj.toolbarModule.refreshToolbarOverflow();
        }, 400);
      }
    }
    function created() {
      textArea = rteObj.contentModule.getEditPanel();
    }
    return (<RichTextEditorComponent ref={ (richtexteditor) => { rteObj =
richtexteditor; }} height={450} toolbarSettings={toolbarSettings}
showCharCount={true} actionComplete={actionComplete} =
actionComplete.bind(this)} created={created} = created.bind(this)}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>

```

```

        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]}/>
    </RichTextEditorComponent>);
  }
  export default App;
  {% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - CodeView sample
 */
import { createElement } from '@syncfusion/ej2-base';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import CodeMirror from 'codemirror';
import 'codemirror/mode/css/css.js';
import 'codemirror/mode/htmlmixed/htmlmixed.js';
import 'codemirror/mode/javascript/javascript';
import * as React from 'react';
import { useEffect } from "react";
function App() {
  useEffect(() => {
    return () => {
      <div>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary
functionality on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>

```

```

        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
    </div>;
  };
});
let myCodeMirror: any;
let textArea: HTMLInputElement;
let rteObj: RichTextEditorComponent;
let toolbarSettings: object = {
  items: ['SourceCode']
}
function value(): JSX.Element {
  return(
  );
};
function mirrorConversion(e: any): void {
  const id: string = rteObj.getID() + 'mirror-view';
  let mirrorView: HTMLInputElement = rteObj.element.querySelector('#' + id) as
HTMLInputElement;
  const charCount: HTMLInputElement = rteObj.element.querySelector('.e-rte-
character-count') as HTMLInputElement;
  if (e.targetItem === 'Preview') {
    textArea.style.display = 'block';
    mirrorView.style.display = 'none';
    textArea.innerHTML = myCodeMirror.getValue();
    charCount.style.display = 'block';
  } else {
    if (!mirrorView) {
      mirrorView = createElement('div', { className: 'e-content' });
      mirrorView.id = id;
      (textArea as any).parentNode.appendChild(mirrorView);
    } else {
      mirrorView.innerHTML = '';
    }
    textArea.style.display = 'none';
    mirrorView.style.display = 'block';
    renderCodeMirror(mirrorView, rteObj.value);
    charCount.style.display = 'none';
  }
}

```

```

    }
  }
  function renderCodeMirror(mirrorView: HTMLElement, content: string): void {
    myCodeMirror = CodeMirror(mirrorView, {
      lineNumbers: true,
      lineWrapping: true,
      mode: 'text/html',
      value: content
    });
  }
  function actionComplete(e: any) {
    textArea = (rteObj as any).contentModule.getEditPanel() as HTMLElement;
    if (e.targetItem && (e.targetItem === 'SourceCode' || e.targetItem ===
'Preview')) {
      (rteObj.sourceCodeModule.getPanel() as any).style.display = 'none';
      mirrorConversion(e);
    } else {
      setTimeout(() => {
        rteObj.toolbarModule.refreshToolbarOverflow();
      }, 400);
    }
  }
  function created(): void {
    textArea = (rteObj as any).contentModule.getEditPanel() as HTMLElement;
  }
  return (
    <RichTextEditorComponent ref={ (richtexteditor) => { rteObj =
richtexteditor! } } height={450} toolbarSettings={toolbarSettings}
showCharCount={true} actionComplete={actionComplete
=actionComplete.bind(this)} created={created = created.bind(this)}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
      </ul>
    </RichTextEditorComponent>
  )

```

```

        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]} />
  </RichTextEditorComponent>
);
}
export default App;
{% endraw %}

```

Undo/Redo manager

Undo and redo tools allow you to edit the text by disregard/cancel the recently made changes and restore it to previous state. It is a useful tool to restore the performed action which got changed by mistake. By default, upto 30 actions can be undo/redo in the editor.

To undo and redo operations, do one of the following:

- Press the undo/redo button on the toolbar.
- Press the **Ctrl + Z/Ctrl + Y** combination on the keyboard.

Customize the undo/redo step count using the [undoRedoSteps](#) property. By default, undo/redo actions take 300ms time interval for store the action to the undoRedoManager. The time interval can be customized by using the [undoRedoTimer](#).

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - Undo/Redo Manager Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['Undo', 'Redo']
  };
  render() {
    return (<div>
      <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings} undoRedoSteps={50}
undoRedoTimer={400}>

```

```

    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
    you get") editor that provides the best user experience to create and update
    the content.
      Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>
  </RichTextEditorComponent>
</div>);
}
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Undo/Redo Manager Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';

```

```

class App extends React.Component<{}, {}> {
  private toolbarSettings: object = {
    items: ['Undo', 'Redo']
  }
  public render() {
    return (<div>
      <RichTextEditorComponent height={450}
        toolbarSettings={this.toolbarSettings} undoRedoSteps={50}
        undoRedoTimer={400}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
        you get") editor that provides the best user experience to create and update
        the content.
          Users can format their content using standard toolbar commands.</p>
          <p><b>Key features:</b></p>
          <ul>
            <li>
              <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
              <p>Capable of handling markdown editing.</p>
            </li>
            <li>
              <p>Contains a modular library to load the necessary functionality
on demand.</p>
            </li>
            <li>
              <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
              <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
              <p>Supports third-party library integration.</p>
            </li>
            <li>
              <p>Allows preview of modified content before saving it.</p>
            </li>
            <li>
              <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
            </li>
            <li>
              <p>Contains undo/redo manager.</p>
            </li>
            <li>
              <p>Creates bulleted and numbered lists.</p>
            </li>
          </ul>
          <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
        </RichTextEditorComponent>
      </div>);
  }
}
export default App;

```


[Functional-component]**APP.JSX**

```

/**
 * Rich Text Editor - Undo/Redo Manager Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['Undo', 'Redo']
  };
  return (<div>
    <RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
undoRedoSteps={50} undoRedoTimer={400}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>

```

```

        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
  </RichTextEditorComponent>
</div>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Undo/Redo Manager Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings: object = {
    items: ['Undo', 'Redo']
  }
  return (<div>
    <RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings} undoRedoSteps={50} undoRedoTimer={400}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>

```

```

        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
  </RichTextEditorComponent>
</div>);
}
export default App;

```

Prevention of cross-site scripting (XSS)

The Rich Text Editor allow the users to edit the content with security by preventing cross-site scripting (XSS). By default, provided built-in support to remove the elements from editor content which cause XSS attack. The editor removes the elements based on the attributes if there is possible to execute script.

In the following sample, removed `script` tag and `onmouseover` attribute from content of the Rich Text Editor.

[Class-component]

APP.JSX

```

{% raw %}
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  template = `<div onmouseover='javascript:alert(1)'>Prevention of Cross
Sit Scripting (XSS) </div> <script>alert('hi')</script>`;
  render() {
    return (<div>
      <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) => {
this.rteObj = richtexteditor; }} valueTemplate={this.template}>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
      </RichTextEditorComponent>
    </div>);
  }
}
export default App;
{% endraw %}

```

APP.TSX

```
{% raw %}
```

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{},{}> {
  public rteObj: RichTextEditorComponent;
  public template: string = `<div onmouseover='javascript:alert(1)'>Prevention
of Cross Sit Scripting (XSS) </div> <script>alert('hi')</script>`;
  public render() {
    return (<div>
      <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) => {
this.rteObj = richtexteditor; }} valueTemplate={this.template}>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
      </RichTextEditorComponent>
    </div>);
  }
}
export default App;
{% endraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let template = `<div onmouseover='javascript:alert(1)'>Prevention of
Cross Sit Scripting (XSS) </div> <script>alert('hi')</script>`;
  let rteObj;
  return (<div>
    <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) => {
rteObj = richtexteditor; }} valueTemplate={template}>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>
    </RichTextEditorComponent>
  </div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let template: string = `<div onmouseover='javascript:alert(1)'>Prevention of
Cross Sit Scripting (XSS) </div> <script>alert('hi')</script>`;
  let rteObj: RichTextEditorComponent;
  return (<div>

```

```

    <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) =>
    {rteObj = richtexteditor; }} valueTemplate={template}>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
    />
    </RichTextEditorComponent>
  </div>;
}
export default App;
{% endraw %}

```

It's only applicable to editorMode as HTML.

Custom cross-site scripting

You can also filter the elements and attributes additionally which cause the XSS attack through [beforeSanitizeHtml](#) event. Return the value from the event argument `helper` function to apply in the editor. If you want to prevent the built-in support and make own cross-site scripting rules, set `cancel` argument as true.

The following sample demonstrate how to filter `script` tag from value.

[Class-component]

APP.JSX

```

{% raw %}
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { detach } from '@syncfusion/ej2-base';
import * as React from 'react';
class App extends React.Component {
  template = `<div onmouseover='javascript:alert(1)'>Prevention of Cross
Sit Scripting (XSS) </div> <script>alert('hi')</script>`;
  rteObj;
  beforeSanitizeHtml(e) {
    e.helper = (value) => {
      e.cancel = true;
      let temp = document.createElement('div');
      temp.innerHTML = value;
      let scriptTag = temp.querySelector('script');
      if (scriptTag) {
        detach(scriptTag);
      }
      return temp.innerHTML;
    };
  }
  render() {
    return (<div>
      <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) => {
        this.rteObj = richtexteditor; }} valueTemplate={this.template}
        beforeSanitizeHtml={this.beforeSanitizeHtml} =
        this.beforeSanitizeHtml.bind(this)}>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
      </RichTextEditorComponent>
    </div>);
  }
}

```

```

}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar, BeforeSanitizeHtmlArgs } from
 '@syncfusion/ej2-react-richtexteditor';
import { detach } from '@syncfusion/ej2-base';
import * as React from 'react';
class App extends React.Component<{},{}> {
  public template: string = `<div onmouseover='javascript:alert(1)'>Prevention
of Cross Site Scripting (XSS) </div> <script>alert('hi')</script>`;
  public rteObj: RichTextEditorComponent;
  public beforeSanitizeHtml(e: BeforeSanitizeHtmlArgs) {
    e.helper = (value: string) => {
      e.cancel = true;
      let temp: HTMLElement = document.createElement('div');
      temp.innerHTML = value;
      let scriptTag: HTMLElement = temp.querySelector('script');
      if (scriptTag) {
        detach(scriptTag);
      }
      return temp.innerHTML;
    }
  }
  public render() {
    return (<div>
      <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) => {
this.rteObj = richtexteditor; }} valueTemplate={this.template}
beforeSanitizeHtml={this.beforeSanitizeHtml}
this.beforeSanitizeHtml.bind(this)}>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
      </RichTextEditorComponent>
    </div>);
  }
}
export default App;
{% endraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { detach } from '@syncfusion/ej2-base';
import * as React from 'react';
function App() {
  let template = `<div onmouseover='javascript:alert(1)'>Prevention of
Cross Site Scripting (XSS) </div> <script>alert('hi')</script>`;

```

```

let rteObj;
function beforeSanitizeHtml(e) {
  e.helper = (value) => {
    e.cancel = true;
    let temp = document.createElement('div');
    temp.innerHTML = value;
    let scriptTag = temp.querySelector('script');
    if (scriptTag) {
      detach(scriptTag);
    }
    return temp.innerHTML;
  };
}
return (<div>
  <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) => {
    rteObj = richtexteditor; }} valueTemplate={template}
    beforeSanitizeHtml={beforeSanitizeHtml = beforeSanitizeHtml.bind(this)}>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
  </RichTextEditorComponent>
</div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar, BeforeSanitizeHtmlArgs } from
'@syncfusion/ej2-react-richtexteditor';
import { detach } from '@syncfusion/ej2-base';
import * as React from 'react';
function App() {
  let template: string = `<div onmouseover='javascript:alert(1)'>Prevention of
Cross Site Scripting (XSS) </div> <script>alert('hi')</script>`;
  let rteObj: RichTextEditorComponent;
  function beforeSanitizeHtml(e: BeforeSanitizeHtmlArgs) {
    e.helper = (value: string) => {
      e.cancel = true;
      let temp: HTMLElement = document.createElement('div');
      temp.innerHTML = value;
      let scriptTag: HTMLElement = temp.querySelector('script');
      if (scriptTag) {
        detach(scriptTag);
      }
      return temp.innerHTML;
    }
  }

  return (<div>
    <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) => {rteObj
= richtexteditor; }} valueTemplate={template}
    beforeSanitizeHtml={beforeSanitizeHtml = beforeSanitizeHtml.bind(this)}>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  </div>);
}

```

```

}
export default App;
{% enddraw %}

```

Resizable support

This feature allows the editor to be resized dynamically. The users can enable or disable this feature using the `enableResize` property in the Rich Text Editor. If `enableResize` is set to true, the Rich Text Editor component creates grip at the bottom right corner, which allows resizing the component in the diagonal direction. The following sample demonstrates the resizable feature.

Enabling the resizable support

To render the Rich Text Editor in the resizable mode, set the `enableResize` property to true. The above feature is built as module and hence the `Resize` module needs to be included in your application.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - Resizable Sample
 */
import { HtmlEditor, Image, Inject, Link, Resize, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  resize = true;
  render() {
    return (
      <RichTextEditorComponent enableResize={this.resize}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <Inject services={[HtmlEditor, Toolbar, Image, Link, Resize,
QuickToolbar]}/>
      </RichTextEditorComponent>
    );
  }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Resizable Sample
 */
import { HtmlEditor, Image, Inject, Link, Resize, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  public resize: boolean = true;
  public render() {
    return (
      <RichTextEditorComponent enableResize={this.resize} >

```



```

    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <Inject services={[HtmlEditor, Toolbar, Image, Link, Resize,
QuickToolbar]} />
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Resizable Sample
 */
import { HtmlEditor, Image, Inject, Link, Resize, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let resize = true;
  return (<RichTextEditorComponent enableResize={resize}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar
commands.</p>
    <Inject services={[HtmlEditor, Toolbar, Image, Link, Resize,
QuickToolbar]} />
    </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Resizable Sample
 */
import { HtmlEditor, Image, Inject, Link, Resize, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let resize: boolean = true;
  return (
    <RichTextEditorComponent enableResize={resize} >
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar
commands.</p>

```

```

        <Inject services={[HtmlEditor, Toolbar, Image, Link, Resize,
QuickToolbar]} />
        </RichTextEditorComponent>
    );
}
export default App;

```

Specifying the Minimum and Maximum width and height for Resize

To have a restricted resizable area for the RichTextEditor, you need to specify the min-width, max-width, min-height, and max-height CSS properties for the control's wrapper element. By default, the control is capable of resizing upto the current viewport. The `e-richtexteditor` CSS class will be available in the component's wrapper and can be used for applying the above mentioned styles.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - Resizable Sample
 */
import { HtmlEditor, Image, Inject, Link, Resize, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
    resize = true;
    render() {
        return (<RichTextEditorComponent enableResize={this.resize}>
            <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
            Users can format their content using standard toolbar commands.</p>
            <Inject services={[HtmlEditor, Toolbar, Image, Link, Resize,
QuickToolbar]} />
            </RichTextEditorComponent>);
    }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Resizable Sample
 */
import { HtmlEditor, Image, Inject, Link, Resize, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    public resize: boolean = true;
    public render() {
        return (
            <RichTextEditorComponent enableResize={this.resize} >

```

```

    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <Inject services={[HtmlEditor, Toolbar, Image, Link, Resize,
QuickToolbar]} />
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Resizable Sample
 */
import { HtmlEditor, Image, Inject, Link, Resize, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let resize = true;
  return (<RichTextEditorComponent enableResize={resize}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <Inject services={[HtmlEditor, Toolbar, Image, Link, Resize,
QuickToolbar]} />
    </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Resizable Sample
 */
import { HtmlEditor, Image, Inject, Link, Resize, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let resize: boolean = true;
  return (
    <RichTextEditorComponent enableResize={resize} >
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <Inject services={[HtmlEditor, Toolbar, Image, Link, Resize,
QuickToolbar]} />
    </RichTextEditorComponent>
  );
}

```

```
);
}
export default App;
```

Number and Bullet Format Lists

This feature allows the user to change the appearance of the Numbered and Bulleted lists. Users can also apply different numbering or bullet formats lists such as lowercase greek, upper Alpha, square and circles. You can also customize the style type of the lists to be populated in the dropdown from the toolbar by using the `numberFormatList` and `bulletFormatList` properties in the Rich Text Editor.

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - Number/Bullet Format list Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['NumberFormatList', 'BulletFormatList']
  };
  render() {
    return (
      <div>
        <RichTextEditorComponent toolbarSettings={this.toolbarSettings}>
          <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
          Users can format their content using standard toolbar commands.</p>
          <p><b>Key features:</b></p>
          <ul>
            <li>
              <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
              <p>Capable of handling markdown editing.</p>
            </li>
            <li>
              <p>Contains a modular library to load the necessary functionality
on demand.</p>
            </li>
            <li>
              <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
              <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
              <p>Supports third-party library integration.</p>
            </li>
          </ul>
        </RichTextEditorComponent>
      </div>
    );
  }
}
```

```

        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
  </RichTextEditorComponent>
</div>);
}
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Number/Bullet Format list Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private toolbarSettings: object = {
    items: ['NumberFormatList', 'BulletFormatList']
  }
  public render() {
    return <div>
      <RichTextEditorComponent toolbarSettings={this.toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>

```

```

        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
  </RichTextEditorComponent>
</div>);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Number/Bullet Format list Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['NumberFormatList', 'BulletFormatList']
  };
  return (<div>
    <RichTextEditorComponent toolbarSettings={toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>

```

```

        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
    </RichTextEditorComponent>
  </div>;
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Number/Bullet Format list Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings: object = {
    items: ['NumberFormatList', 'BulletFormatList']
  }
  return (<div>
    <RichTextEditorComponent toolbarSettings={toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>

```

```

        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
  </RichTextEditorComponent>
</div>);
}
export default App;

```

Xhtml validation in React Rich text editor component

The editor provides an option to validate the source content of the Rich Text Editor against the XHTML standard using the 'enableXhtml' property. When you enter or modify content in the editor, it continuously checks the XHTML source content and removes elements and attributes that are not valid.

The editor checks the following settings on validation:

Attributes

- Must be specified in lowercase.
- Proper use of quotation marks around the attributes.
- Must be valid attributes for corresponding HTML element.
- All the required attributes must be included in the HTML element.

HTML Elements

- Must be in lowercase.
- All opening tags must be closed.
- Allows only the valid HTML elements.
- Elements must be properly nested.
- All elements must have one root element.
- Should not use inline elements inside the block elements.

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - RTL Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  render() {
    return (<RichTextEditorComponent height={450} enableXhtml={true}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </ul>
    </RichTextEditorComponent>
  )
}
```

```

        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
</RichTextEditorComponent>);
    }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - RTL Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    public render() {
        return (
            <RichTextEditorComponent height={450} enableXhtml={true}>
                <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
                Users can format their content using standard toolbar commands.</p>
                <p><b>Key features:</b></p>
                <ul>
                    <li>
                        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
                    </li>
                    <li>
                        <p>Capable of handling markdown editing.</p>
                    </li>
                    <li>
                        <p>Contains a modular library to load the necessary functionality
on demand.</p>
                    </li>
                    <li>
                        <p>Provides a fully customizable toolbar.</p>
                    </li>
                    <li>
                        <p>Provides HTML view to edit the source directly for
developers.</p>
                    </li>
                    <li>
                        <p>Supports third-party library integration.</p>
                    </li>
                    <li>
                        <p>Allows preview of modified content before saving it.</p>
                    </li>
                </ul>
            </RichTextEditorComponent>
        );
    }
}

```

```

        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
</RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - RTL Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    return (<RichTextEditorComponent height={450} enableXhtml={true}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
                <p>Capable of handling markdown editing.</p>
            </li>
            <li>
                <p>Contains a modular library to load the necessary functionality
on demand.</p>
            </li>
            <li>
                <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
                <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>

```

```

        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
  </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - RTL Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  return (
    <RichTextEditorComponent height={450} enableXhtml={true}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
      </ul>
    </RichTextEditorComponent>
  );
}

```

```

        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
export default App;

```

Inline mode in React Rich text editor component

This is the inline example for the Rich Text Editor. For this you must set the [inlineMode](#) property.

Inline edition allows you to select any editable element or click the element on the page and edit it in-place.

Inline editing is a true WYSIWYG formation and on the contrary to Rich Text Editor HTML/MD editing, the styles that are used for edited content comes directly from the document stylesheet. This means that inline editors ignore the default Rich Text Editor content styles.

Show on select/click

Enabling the [onSelection](#) option of inlineMode makes the inline Rich Text Editor to appear. You can select the text in the editable area otherwise the inline Rich Text Editor will be appear once click into the editable area.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - InlineMode Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  inlineMode = {
    enable: true,
    onSelection: true
  };
  toolbarSettings = {

```

```

        items: [
            'Bold', 'Italic', 'Underline', 'Formats', '-',
            'Alignments', 'OrderedList', 'UnorderedList', 'CreateLink'
        ]
    };
    render() {
        return (
            <RichTextEditorComponent height={450}
            inlineMode={this.inlineMode} toolbarSettings={this.toolbarSettings}>
                <p>The Rich Text Editor component is WYSIWYG ("what you see is what
                you get") editor that provides the best user experience to create and update
                the content. Users can format their content using standard toolbar
                commands.</p>
                <p><b>Key features:</b></p>
                <ul>
                    <li>
                        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
                    </li>
                    <li>
                        <p>Capable of handling markdown editing.</p>
                    </li>
                    <li>
                        <p>Contains a modular library to load the necessary functionality
                        on demand.</p>
                    </li>
                    <li>
                        <p>Provides a fully customizable toolbar.</p>
                    </li>
                    <li>
                        <p>Provides HTML view to edit the source directly for
                        developers.</p>
                    </li>
                    <li>
                        <p>Supports third-party library integration.</p>
                    </li>
                    <li>
                        <p>Allows preview of modified content before saving it.</p>
                    </li>
                    <li>
                        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
                        etc.</p>
                    </li>
                    <li>
                        <p>Contains undo/redo manager.</p>
                    </li>
                    <li>
                        <p>Creates bulleted and numbered lists.</p>
                    </li>
                </ul>
                <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>
            </RichTextEditorComponent>
        );
    }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - InlineMode Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}> {
  private inlineMode: object = {
    enable: true,
    onSelection: true
  }
  private toolbarSettings: object = {
    items: [
      'Bold', 'Italic', 'Underline', 'Formats', '-',
      'Alignments', 'OrderedList', 'UnorderedList', 'CreateLink'
    ]
  }
  public render() {
    return (
      <RichTextEditorComponent height={450} inlineMode={this.inlineMode}
toolbarSettings={this.toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
          </li>
          <li>
            <p>Contains undo/redo manager.</p>
          </li>
        </ul>
      </RichTextEditorComponent>
    );
  }
}

```

```

        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
    />
  </RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - InlineMode Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  let inlineMode = {
    enable: true,
    onSelection: true
  };
  let toolbarSettings = {
    items: [
      'Bold', 'Italic', 'Underline', 'Formats', '-',
      'Alignments', 'OrderedList', 'UnorderedList', 'CreateLink'
    ]
  };
  return (
    <RichTextEditorComponent height={450} inlineMode={inlineMode}
    toolbarSettings={toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>

```



```

        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
  </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - InlineMode Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let inlineMode: object = {
    enable: true,
    onSelection: true
  }
  let toolbarSettings: object = {
    items: [
      'Bold', 'Italic', 'Underline', 'Formats', '-',
      'Alignments', 'OrderedList', 'UnorderedList', 'CreateLink'
    ]
  }
  return (
    <RichTextEditorComponent height={450} inlineMode={inlineMode}
toolbarSettings={toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content. Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>

```

```

        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
  </RichTextEditorComponent>
);
}
export default App;

```

Paste cleanup in React Rich text editor component

The Rich Text Editor allows you to reduce the effort while converting the Microsoft Word content to HTML format with format and styles.

MS Word to HTML

By default, Rich Text Editor consider the following processes on paste content from Microsoft Word.

List conversion: The list elements copied from the Microsoft Word document contains paragraph tags with styles and classes. The list elements are converted to standard HTML list elements by referring the styles and class names in the paragraph tags.

Converting style: The styles of the elements copied from the Microsoft Word document are converted to standard CSS styles and added as inline styles for each respective element.

Tags and comments: The Microsoft Word specific XML tags and comments are removed when cleanup on paste.

Paste cleanup

You can control the formatting and styles on pasting the content to the editor using the pasteCleanup settings property. The following settings are available to clean up the content:

API	Description	Default Value	Type
prompt	To invoke prompt dialog with paste options on pasting the content in editor.	false	boolean
plainText	To paste the content as plain text.	false	boolean
keepFormat	To keep the same format with copied content.	true	boolean
deniedTags	To ignore the tags when pasting HTML content.	null	string[]
deniedAttrs	To paste the content by filtering out these attributes from the content.	null	string[]
allowedStyleProps	To paste the content by accepting these style attributes and removing other style attributes.	['background', 'background-color', 'border', 'border-bottom', 'border-left', 'border-radius', 'border-right', 'border-style', 'border-top', 'border-width', 'clear', 'color', 'cursor', 'direction', 'display', 'float', 'font', 'font-family', 'font-size', 'font-weight', 'font-style', 'height', 'left', 'line-height', 'margin', 'margin-top', 'margin-left', 'margin-right', 'margin-bottom', 'max-height', 'max-width', 'min-height', 'min-width', 'overflow', 'overflow-x', 'overflow-y', 'padding', 'padding-bottom', 'padding-left', 'padding-right', 'padding-top', 'position', 'right', 'table-layout', 'text-align', 'text-decoration', 'text-indent', 'top', 'vertical-align', 'visibility', 'white-space', 'width']	string[]

Rich Text Editor features are segregated into individual feature-wise modules. To use paste cleanup, inject paste cleanup module using the `RichTextEditor.Inject(PasteCleanup)`.

Prompt dialog

When `prompt` is set to true, pasting the content in the editor will open a dialog box that contains three options `Keep`, `Clean`, and `Plain Text` as radio buttons:

1. `Keep`: Radio button to keep the same format with copied content.
2. `Clean`: Radio button to clear all the style formats with copied content.
3. `Plain Text`: Radio button to paste the copied content as plain text without any formatting or style (including the removal of all tags).

When `prompt` value is set true, the API properties `plainText` and `keepFormat` will not be considered for processing when pasting the content.

Paste as plain text

When `plainText` is set to true, the copied content will be converted as plain text by removing all the HTML tags and styles applied to it and only the plain text is pasted in the editor.

When `plainText` value is set true, the API property `prompt` should be set to false, and `keepFormat` will not be considered for processing when pasting the content.

Keep format

When `keepFormat` is set to true, the copied content will maintain all the style formatting allowed in the `allowedStyleProps` on pasting the content in the editor.

When `keepFormat` is set to false, the style in the copied content will be removed without considering the allowed styles in the `allowedStyleProps` when pasting the content in the editor.

When `keepFormat` value is set true, the API property `prompt` and `plainText` should be set to false.

Denied tags

When `deniedTags` values are set, the tags that matches the 'denied tags' list will be removed on pasting the copied content in the editor. For Example,

1. `'a'`: Paste the content by filtering out anchor tags.
2. `'a[!href]'`: Paste the content by filtering out anchor tags that do not have the 'href' attribute.
3. `'a[href, target]'`: Paste the content by filtering out anchor tags that have the 'href' and 'target' attributes.

Denied attributes

When the `deniedAttrs` values are set, the attributes that matches the 'denied attributes' list will be removed on pasting the copied content in the editor. For Example,

`'id', 'title'`: This will remove the attributes 'id' and 'title' from all tags.

Allowed style properties

By default, the following basic styles are allowed on pasting the content to the editor.

`['background', 'background-color', 'border', 'border-bottom', 'border-left', 'border-radius', 'border-right', 'border-style', 'border-top', 'border-width', 'clear', 'color', 'cursor', 'direction', 'display', 'float', 'font', 'font-family', 'font-size', 'font-weight', 'font-style', 'height', 'left', 'line-height', 'margin', 'margin-top', 'margin-left', 'margin-right', 'margin-bottom', 'max-height', 'max-width', 'min-height', 'min-width', 'overflow', 'overflow-x', 'overflow-y', 'padding', 'padding-bottom', 'padding-left', 'padding-right', 'padding-top', 'position', 'right', 'table-layout', 'text-align', 'text-decoration', 'text-indent', 'top', 'vertical-align', 'visibility', 'white-space', 'width']`

When you configure `allowedStyleProps`, the styles, which matches the 'allowed style properties' list are allowed, all other style properties will be removed on pasting the content in the editor.

For Example,

`allowedStyleProps: ['color', 'margin']`: This will allow only the style properties 'color' and 'margin' in each pasted element.

In the following example, the paste cleanup related settings are explained with its module configuration

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - Paste Cleanup Sample
 */
import { HtmlEditor, Image, Inject, Link, PasteCleanup, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|'],
  };
}
```

```

        'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
        'SourceCode', 'FullScreen', '|', 'Undo', 'Redo'
    ]
};
pasteCleanupSettings = {
    allowedStyleProps: ['color', 'margin', 'font-size'],
    deniedAttrs: ['class', 'title', 'id'],
    deniedTags: ['a'],
    keepFormat: false,
    plainText: false,
    prompt: true
};
render() {
    return (<RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}
pasteCleanupSettings={this.pasteCleanupSettings}>
    <p>Rich Text Editor is a WYSIWYG editing control which will reduce
the effort for users while trying to express their formatting word content as
HTML or Markdown format.</p>
    <p><b>Paste Cleanup properties:</b></p>
    <ul>
        <li>
            <p>prompt - specifies whether to enable the prompt when
pasting in Rich Text Editor.</p>
        </li>
        <li>
            <p>plainText - specifies whether to paste as plain text or
not in Rich Text Editor.</p>
        </li>
        <li>
            <p>keepFormat- specifies whether to keep or remove the format
when pasting in Rich Text Editor.</p>
        </li>
        <li>
            <p>deniedTags - specifies the tags to restrict when pasting
in Rich Text Editor.</p>
        </li>
        <li>
            <p>deniedAttributes - specifies the attributes to restrict
when pasting in Rich Text Editor.</p>
        </li>
        <li>
            <p>allowedStyleProperties - specifies the allowed style
properties when pasting in Rich Text Editor.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
PasteCleanup]}/>
    </RichTextEditorComponent>);
}
}
export default App;

```

APP.TSX

```
/**
```

```

* Rich Text Editor - Paste Cleanup Sample
*/
import { HtmlEditor, Image, Inject, Link, PasteCleanup, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{},{}> {
  private toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo'
    ]
  }
  private pasteCleanupSettings: object = {
    allowedStyleProps: ['color', 'margin', 'font-size'],
    deniedAttrs: ['class', 'title', 'id'],
    deniedTags: ['a'],
    keepFormat: false,
    plainText: false,
    prompt: true
  }
  public render() {
    return (
      <RichTextEditorComponent height={450}
toolbarSettings={this.toolbarSettings}
pasteCleanupSettings={this.pasteCleanupSettings}>
        <p>Rich Text Editor is a WYSIWYG editing control which will reduce
the effort for users while trying to express their formatting word content as
HTML or Markdown format.</p>
        <p><b>Paste Cleanup properties:</b></p>
        <ul>
          <li>
            <p>prompt - specifies whether to enable the prompt when
pasting in Rich Text Editor.</p>
          </li>
          <li>
            <p>plainText - specifies whether to paste as plain text or
not in Rich Text Editor.</p>
          </li>
          <li>
            <p>keepFormat- specifies whether to keep or remove the format
when pasting in Rich Text Editor.</p>
          </li>
          <li>
            <p>deniedTags - specifies the tags to restrict when pasting
in Rich Text Editor.</p>
          </li>
          <li>
            <p>deniedAttributes - specifies the attributes to restrict
when pasting in Rich Text Editor.</p>
          </li>
        </ul>
      </RichTextEditorComponent>
    );
  }
}

```

```

        <p>allowedStyleProperties - specifies the allowed style
        properties when pasting in Rich Text Editor.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
    PasteCleanup]} />
  </RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Paste Cleanup Sample
 */
import { HtmlEditor, Image, Inject, Link, PasteCleanup, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
    'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
    'LowerCase', 'UpperCase', '|',
    'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
    'Outdent', 'Indent', '|',
    'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
    'SourceCode', 'FullScreen', '|', 'Undo', 'Redo'
  ]
};
  let pasteCleanupSettings = {
    allowedStyleProps: ['color', 'margin', 'font-size'],
    deniedAttrs: ['class', 'title', 'id'],
    deniedTags: ['a'],
    keepFormat: false,
    plainText: false,
    prompt: true
  };
  return (<RichTextEditorComponent height={450}
toolbarSettings={toolbarSettings}
pasteCleanupSettings={pasteCleanupSettings}>
    <p>Rich Text Editor is a WYSIWYG editing control which will reduce the
    effort for users while trying to express their formatting word content as
    HTML or Markdown format.</p>
    <p><b>Paste Cleanup properties:</b></p>
    <ul>
      <li>
        <p>prompt - specifies whether to enable the prompt when pasting
in Rich Text Editor.</p>
      </li>
      <li>

```

```

        <p>plainText - specifies whether to paste as plain text or not
in Rich Text Editor.</p>
      </li>
    </li>
    <p>keepFormat- specifies whether to keep or remove the format
when pasting in Rich Text Editor.</p>
    </li>
    </li>
    <p>deniedTags - specifies the tags to restrict when pasting in
Rich Text Editor.</p>
    </li>
    </li>
    <p>deniedAttributes - specifies the attributes to restrict when
pasting in Rich Text Editor.</p>
    </li>
    </li>
    <p>allowedStyleProperties - specifies the allowed style
properties when pasting in Rich Text Editor.</p>
    </li>
  </ul>
  <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
PasteCleanup]}/>
</RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Paste Cleanup Sample
 */
import { HtmlEditor, Image, Inject, Link, PasteCleanup, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo'
  ]
}
let pasteCleanupSettings: object = {
  allowedStyleProps: ['color', 'margin', 'font-size'],
  deniedAttrs: ['class', 'title', 'id'],
  deniedTags: ['a'],
  keepFormat: false,
  plainText: false,
  prompt: true
}
return (

```



```

<RichTextEditorComponent height={450} toolbarSettings={toolbarSettings}
pasteCleanupSettings={pasteCleanupSettings}>
  <p>Rich Text Editor is a WYSIWYG editing control which will reduce the
  effort for users while trying to express their formatting word content as
  HTML or Markdown format.</p>
  <p><b>Paste Cleanup properties:</b></p>
  <ul>
    <li>
      <p>prompt - specifies whether to enable the prompt when pasting
in Rich Text Editor.</p>
    </li>
    <li>
      <p>plainText - specifies whether to paste as plain text or not
in Rich Text Editor.</p>
    </li>
    <li>
      <p>keepFormat- specifies whether to keep or remove the format
      when pasting in Rich Text Editor.</p>
    </li>
    <li>
      <p>deniedTags - specifies the tags to restrict when pasting in
      Rich Text Editor.</p>
    </li>
    <li>
      <p>deniedAttributes - specifies the attributes to restrict when
      pasting in Rich Text Editor.</p>
    </li>
    <li>
      <p>allowedStyleProperties - specifies the allowed style
      properties when pasting in Rich Text Editor.</p>
    </li>
  </ul>
  <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
  PasteCleanup]} />
</RichTextEditorComponent>
);
}
export default App;

```

Mention integration in React Rich text editor component

By integrating the [Mention](#) component with a Rich Text Editor, users can easily mention or tag other users or objects from the suggested list without having to manually type out their names or other identifying information.

The [target](#) property of the Mention component allows you to specify the ID of the content editable div element within the Rich Text Editor that you want to bind the Mention component to. This allows you to enable the Mention functionality within the Rich Text Editor, so that users can mention or tag other users or objects from the suggested list while editing the text.

When the user types the @ symbol followed by a character, the Rich Text Editor will display a list of suggestions for items that the user can select from. The user can then select an item from the list by clicking on it, or by typing the name of the item they want to tag.

In the following sample, configured the following properties with popup dimensions.

- [allowSpaces](#) - Allow to continue search action if user enter space after mention character while searching.
- [suggestionCount](#) - The maximum number of items that will be displayed in the suggestion list.
- [itemTemplate](#) - Used to display the customized appearance in suggestion list.

[Class-component]

APP.JSX

```
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import { MentionComponent } from '@syncfusion/ej2-react-dropdowns';
import * as React from 'react';
class App extends React.Component {
  data = [
    { Name: "Selma Rose", Status: "active", EmployeeImage: "2.png",
EmailId: "selma@gmail.com" },
    { Name: "Maria", Status: "active", EmployeeImage: "1.png", EmailId:
"maria@gmail.com" },
    { Name: "Russo Kay", Status: "busy", EmployeeImage: "8.png", EmailId:
"russo@gmail.com" },
    { Name: "Camden Kate", Status: "active", EmployeeImage: "9.png",
EmailId: "camden@gmail.com" },
    { Name: "Robert", Status: "busy", EmployeeImage: "dp.png", EmailId:
"robert@gmail.com" },
    { Name: "Garth", Status: "active", EmployeeImage: "7.png", EmailId:
"garth@gmail.com" },
    { Name: "Andrew James", Status: "away", EmployeeImage: "pic04.png",
EmailId: "noah@gmail.com" },
    { Name: "Olivia", Status: "busy", EmployeeImage: "5.png", EmailId:
"olivia@gmail.com" },
    { Name: "Sophia", Status: "away", EmployeeImage: "6.png", EmailId:
"sophia@gmail.com" },
    { Name: "Margaret", Status: "active", EmployeeImage: "3.png",
EmailId: "margaret@gmail.com" },
    { Name: "Ursula Ann", Status: "active", EmployeeImage: "dp.png",
EmailId: "ursula@gmail.com" },
    { Name: "Laura Grace", Status: "away", EmployeeImage: "4.png",
EmailId: "laura@gmail.com" },
    { Name: "Albert", Status: "active", EmployeeImage: "pic03.png",
EmailId: "albert@gmail.com" },
    { Name: "William", Status: "away", EmployeeImage: "8.png", EmailId:
"william@gmail.com" }
  ];
  fieldsData = { text: 'Name' };
  itemTemplate(data) {
    return (<table>
      <tr>
        <td>
          <div id="mention-TemplateList">
            <img className="mentionEmpImage" src={"src/rich-text-
editor/images/" + data.EmployeeImage}/>
            <span className={"e-badge e-badge-success e-badge-overlap e-
badge-dot e-badge-bottom" + data.Status}></span>
          </div>

```

```

        </td>
        <td className="mentionNameList">
            <span className="person">{data.Name}</span>
            <span className="email">{data.EmailId}</span>
        </td>
    </tr>
</table>);
    }
    displayTemplate(data) {
        return <React.Fragment>
            <a href={`mailto:${data.EmailId}`}
title={data.EmailId}>@{data.Name}</a>
        </React.Fragment>;
    }
    actionBeginHandler(args) {
        if (args.requestType === 'EnterAction') {
            args.cancel = true;
        }
    }
    render() {
        return <div className='control-pane'>
            <div className='control-section' id="rte">
                <div className='rte-control-section'>
                    <RichTextEditorComponent id="mention_integration"
placeholder="Type @ and tag the name"
actionBegin={this.actionBeginHandler.bind(this)}>
                        <p>Hello <span contentEditable={false} className='e-mention-
chip'><a href="mailto:maria@gmail.com"
title="maria@gmail.com">@Maria</a></span> &#8203;</p>
                        <p>Welcome to the mention integration with rich text editor demo.
Type <code>@</code> character and tag user from the suggestion list. </p>
                        <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]}>
                            <RichTextEditorComponent>
                                </div>
                            </div>
                            <MentionComponent id="mentionEditor"
target="#mention_integration_rte-edit-view" suggestionCount={8}
showMentionChar={false} allowSpaces={true} dataSource={this.data}
fields={this.fieldsData} popupWidth="250px" popupHeight="200px"
itemTemplate={this.itemTemplate}
displayTemplate={this.displayTemplate}></MentionComponent>
                                </div>);
                            }
                        }
                    }
                export default App;

```

APP.TSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { MentionComponent } from '@syncfusion/ej2-react-dropdowns';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    public mentionObj : MentionComponent;

```

```

private data: { [key: string]: Object }[] = [
  { Name: "Selma Rose", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/2.png",
    EmailId: "selma@gmail.com" },
  { Name: "Maria", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/1.png",
    EmailId: "maria@gmail.com" },
  { Name: "Russo Kay", Status: "busy", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/8.png",
    EmailId: "russo@gmail.com" },
  { Name: "Camden Kate", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/9.png",
    EmailId: "camden@gmail.com" },
  { Name: "Robert", Status: "busy", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/dp.png",
    EmailId: "robert@gmail.com" },
  { Name: "Garth", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/7.png",
    EmailId: "garth@gmail.com" },
  { Name: "Andrew James", Status: "away", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/pic04.png",
    EmailId: "andrew@gmail.com" },
  { Name: "Olivia", Status: "busy", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/5.png",
    EmailId: "olivia@gmail.com" },
  { Name: "Sophia", Status: "away", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/6.png",
    EmailId: "sophia@gmail.com" },
  { Name: "Margaret", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/3.png",
    EmailId: "margaret@gmail.com" },
  { Name: "Ursula Ann", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/dp.png",
    EmailId: "ursula@gmail.com" },
  { Name: "Laura Grace", Status: "away", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/4.png",
    EmailId: "laura@gmail.com" },
  { Name: "Albert", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/pic03.png",
    EmailId: "albert@gmail.com" },
  { Name: "William", Status: "away", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/8.png",
    EmailId: "william@gmail.com" }
];
private fieldsData: { [key: string]: string }={ text: 'Name' };
private itemTemplate(data: any): JSX.Element {
  return (
    <table>
      <tr>
        <td>
          <div id="mention-TemplateList">
            <img className="mentionEmpImage" src={data.EmployeeImage} />
            <span className={"e-badge e-badge-success e-badge-overlap e-
            badge-dot e-badge-bottom"+ data.Status}></span>
          </div>
        </td>
        <td className="mentionNameList">

```

```

        <span className="person">{data.Name}</span>
        <span className="email">{data.EmailId}</span>
      </td>
    </tr>
  </table>
);
}
private displayTemplate(data: any): JSX.Element {
  return (
    <React.Fragment>
      <a href={`mailto:${data.EmailId}`}
title={data.EmailId}>@{data.Name}</a>
    </React.Fragment>
  );
}
public actionBeginHandler(args: any): void {
  if (args.requestType === 'EnterAction' &&
this.mentionObj.element.classList.contains('e-popup-open')) {
    args.cancel = true;
  }
}
render() {
  return (
    <div className='control-pane'>
      <div className='control-section' id="rte">
        <div className='rte-control-section'>
          <RichTextEditorComponent id="mention_integration"
placeholder="Type @ and tag the name"
actionBegin={this.actionBeginHandler.bind(this)} >
            <p>Hello <span contentEditable={false} className='e-mention-
chip'><a href="mailto:maria@gmail.com">
title="maria@gmail.com">@Maria</a></span>&#8203;</p>
            <p>Welcome to the mention integration with rich text editor demo.
Type <code>@</code> character and tag user from the suggestion list. </p>
            <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]} />
          </RichTextEditorComponent>
        </div>
      </div>
      <MentionComponent ref={(scope) => { this.mentionObj = scope; }}
id="mentionEditor" target="#mention_integration_rte-edit-view"
suggestionCount={8} showMentionChar={false} allowSpaces={true}
dataSource={this.data} fields={this.fieldsData} popupWidth="250px"
popupHeight="200px" itemTemplate={this.itemTemplate}
displayTemplate={this.displayTemplate}></MentionComponent>
    </div>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { MentionComponent } from '@syncfusion/ej2-react-dropdowns';
import * as React from 'react';
function App() {
    let data = [
        { Name: "Selma Rose", Status: "active", EmployeeImage: "2.png",
EmailId: "selma@gmail.com" },
        { Name: "Maria", Status: "active", EmployeeImage: "1.png", EmailId:
"maria@gmail.com" },
        { Name: "Russo Kay", Status: "busy", EmployeeImage: "8.png", EmailId:
"russo@gmail.com" },
        { Name: "Camden Kate", Status: "active", EmployeeImage: "9.png",
EmailId: "camden@gmail.com" },
        { Name: "Robert", Status: "busy", EmployeeImage: "dp.png", EmailId:
"robert@gmail.com" },
        { Name: "Garth", Status: "active", EmployeeImage: "7.png", EmailId:
"garth@gmail.com" },
        { Name: "Andrew James", Status: "away", EmployeeImage: "pic04.png",
EmailId: "noah@gmail.com" },
        { Name: "Olivia", Status: "busy", EmployeeImage: "5.png", EmailId:
"olivia@gmail.com" },
        { Name: "Sophia", Status: "away", EmployeeImage: "6.png", EmailId:
"sophia@gmail.com" },
        { Name: "Margaret", Status: "active", EmployeeImage: "3.png",
EmailId: "margaret@gmail.com" },
        { Name: "Ursula Ann", Status: "active", EmployeeImage: "dp.png",
EmailId: "ursula@gmail.com" },
        { Name: "Laura Grace", Status: "away", EmployeeImage: "4.png",
EmailId: "laura@gmail.com" },
        { Name: "Albert", Status: "active", EmployeeImage: "pic03.png",
EmailId: "albert@gmail.com" },
        { Name: "William", Status: "away", EmployeeImage: "8.png", EmailId:
"william@gmail.com" }
    ];
    let fieldsData = { text: 'Name' };
    function itemTemplate(data) {
        return (<table>
            <tr>
                <td>
                    <div id="mention-TemplateList">
                        <img className="mentionEmpImage" src={"src/rich-text-
editor/images/" + data.EmployeeImage}/>
                        <span className={"e-badge e-badge-success e-badge-overlap e-
badge-dot e-badge-bottom" + data.Status}></span>
                    </div>
                </td>
                <td className="mentionNameList">
                    <span className="person">{data.Name}</span>
                    <span className="email">{data.EmailId}</span>
                </td>
            </tr>
        </table>);
    }
    function displayTemplate(data) {
        return (<React.Fragment>

```

```

        <a href={`mailto:${data.EmailId}`}
title={data.EmailId}>@{data.Name}</a>
      </React.Fragment>);
    }
    function actionBeginHandler(args) {
      if (args.requestType === 'EnterAction') {
        args.cancel = true;
      }
    }
    return (<div className='control-pane'>
      <div className='control-section' id="rte">
        <div className='rte-control-section'>
          <RichTextEditorComponent id="mention_integration"
placeholder="Type @ and tag the name"
actionBegin={actionBeginHandler.bind(this)}>
            <p>Hello <span contentEditable={false} className='e-mention-
chip'><a href="mailto:maria@gmail.com"
title="maria@gmail.com">@Maria</a></span>&#8203;</p>
            <p>Welcome to the mention integration with rich text editor demo.
Type <code>@</code> character and tag user from the suggestion list. </p>
            <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]}>
              <RichTextEditorComponent>
                </div>
              </div>
              <MentionComponent id="mentionEditor"
target="#mention_integration_rte-edit-view" suggestionCount={8}
showMentionChar={false} allowSpaces={true} dataSource={data}
fields={fieldsData} popupWidth="250px" popupHeight="200px"
itemTemplate={itemTemplate}
displayTemplate={displayTemplate}></MentionComponent>
            </div>);
  }
  export default App;

```

APP.TSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { MentionComponent } from '@syncfusion/ej2-react-dropdowns';
import * as React from 'react';
function App() {
  let mentionObj;
  let data: { [key: string]: Object }[] = [
    { Name: "Selma Rose", Status: "active", EmployeeImage:
"https://ej2.syncfusion.com/demos/src/rich-text-editor/images/2.png",
EmailId: "selma@gmail.com" },
    { Name: "Maria", Status: "active", EmployeeImage:
"https://ej2.syncfusion.com/demos/src/rich-text-editor/images/1.png",
EmailId: "maria@gmail.com" },
    { Name: "Russo Kay", Status: "busy", EmployeeImage:
"https://ej2.syncfusion.com/demos/src/rich-text-editor/images/8.png",
EmailId: "russo@gmail.com" },

```

```

    { Name: "Camden Kate", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/9.png",
    EmailId: "camden@gmail.com" },
    { Name: "Robert", Status: "busy", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/dp.png",
    EmailId: "robert@gmail.com" },
    { Name: "Garth", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/7.png",
    EmailId: "garth@gmail.com" },
    { Name: "Andrew James", Status: "away", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/pic04.png",
    EmailId: "andrew@gmail.com" },
    { Name: "Olivia", Status: "busy", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/5.png",
    EmailId: "olivia@gmail.com" },
    { Name: "Sophia", Status: "away", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/6.png",
    EmailId: "sophia@gmail.com" },
    { Name: "Margaret", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/3.png",
    EmailId: "margaret@gmail.com" },
    { Name: "Ursula Ann", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/dp.png",
    EmailId: "ursula@gmail.com" },
    { Name: "Laura Grace", Status: "away", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/4.png",
    EmailId: "laura@gmail.com" },
    { Name: "Albert", Status: "active", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/pic03.png",
    EmailId: "albert@gmail.com" },
    { Name: "William", Status: "away", EmployeeImage:
    "https://ej2.syncfusion.com/demos/src/rich-text-editor/images/8.png",
    EmailId: "william@gmail.com" }
  ];
  let fieldsData: { [key: string]: string }={ text: 'Name' };

  function itemTemplate(data: any) {
    return (
      <table>
        <tr>
          <td>
            <div id="mention-TemplateList">
              <img className="mentionEmpImage" src={data.EmployeeImage} />
              <span className={"e-badge e-badge-success e-badge-overlap e-
badge-dot e-badge-bottom"+ data.Status}></span>
            </div>
          </td>
          <td className="mentionNameList">
            <span className="person">{data.Name}</span>
            <span className="email">{data.EmailId}</span>
          </td>
        </tr>
      </table>
    );
  }

  function displayTemplate(data: any) {
    return (

```



```

    <React.Fragment>
      <a href={`mailto:${data.EmailId}`}
title={data.EmailId}>@{data.Name}</a>
    </React.Fragment>
  );
}
function actionBeginHandler(args: any): void {
  if (args.requestType === 'EnterAction' &&
mentionObj.element.classList.contains('e-popup-open')) {
    args.cancel = true;
  }
}
return (
  <div className='control-pane'>
    <div className='control-section' id="rte">
      <div className='rte-control-section'>
        <RichTextEditorComponent id="mention_integration"
placeholder="Type @ and tag the name"
actionBegin={actionBeginHandler.bind(this)} >
          <p>Hello <span contentEditable={false} className='e-mention-
chip'><a href="mailto:maria@gmail.com"
title="maria@gmail.com">@Maria</a></span>&#8203;</p>
          <p>Welcome to the mention integration with rich text editor demo.
Type <code>@</code> character and tag user from the suggestion list. </p>
          <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]} />
        </RichTextEditorComponent>
      </div>
    </div>
    <MentionComponent ref={(scope) => { mentionObj = scope; }}
id="mentionEditor" target="#mention_integration_rte-edit-view"
suggestionCount={8} showMentionChar={false} allowSpaces={true}
dataSource={data} fields={fieldsData} popupWidth="250px" popupHeight="200px"
itemTemplate={itemTemplate}
displayTemplate={displayTemplate}></MentionComponent>
  </div>
);
}
export default App;

```

[View Sample](#)

See Also

- [Mention](#)

Enter key in React Rich text editor component

Rich Text Editor allows to customize the tag that is inserted when pressing the enter key and shift + enter key in the Rich Text Editor.

Enter key customization

By default, the `<p>` tag will be created while pressing the enter key. The enter key can be customized by using the [enterKey](#) property, where the possible tags that can be used to customize are `<p>`, `<div>`, and `
`.

When the enter key is customized with any of the possible values, pressing the enter key in the editor will create a new tag that is configured. Also, when the enter key is configured the default value of the Rich Text Editor will also change respectively with the configured values.

[Class-component]

APP.JSX

```
{% raw %}
/**
 * Rich Text Editor - Enter Key Customization Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  rteValue = `

```

```

        else if (this.enterList.value === 'BR') {
            this.rteObj.enterKey = 'BR';
            this.rteObj.value = `In Rich text Editor, the enter key and shift
+ enter key actions can be customized using the enterKey and shiftEnterKey
APIs. And the possible values are as follows:<ul><li>P - When 'P' is
configured, pressing enter or shift + enter will create a 'p' tag</li><li>DIV
- When 'DIV' is configured, pressing enter or shift + enter will create a
'div' tag</li><li>BR - When 'BR' is configured, pressing enter or shift +
enter will create a 'br' tag</li></ul>`;
        }
    };
    render() {
        return (
            <div className='control-pane'>
                <div className='control-section' id="rte">
                    <div className='rte-control-section'>
                        <table className='api'>
                            <tbody>
                                <tr>
                                    <td>
                                        <div>
                                            <DropDownListComponent id="enterOption"
dataSource={this.enterData} ref={(dropdownlist) => { this.enterList =
dropdownlist; }} fields={this.fields} change={this.enterChange.bind(this)}
value={this.enterValue} popupHeight={this.popupHeight}
placeholder={this.enterPlaceholder} floatLabelType="Always"/>
                                        </div>
                                    </td>
                                </tr>
                            </tbody>
                        </table>
                        <br />
                        <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) =>
{ this.rteObj = richtexteditor; }} height={250} value={this.rteValue}>
                            <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]} />
                        </RichTextEditorComponent>
                    </div>
                </div>
            </div>);
    }
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Enter Key Customization Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { DropDownListComponent, FieldSettingsModel } from '@syncfusion/ej2-
react-dropdowns';
import { createElement } from '@syncfusion/ej2-base';

```

```

import * as React from 'react';
class App extends React.Component<{}, {}> {
  private rteObj: RichTextEditorComponent;
  private rteValue: string = `<p>In Rich text Editor, the enter key and
shift + enter key actions can be customized using the enterKey and
shiftEnterKey APIs. And the possible values are as follows:</p><ul><li>P -
When 'P' is configured, pressing enter or shift + enter will create a 'p'
tag</li><li>DIV - When 'DIV' is configured, pressing enter or shift + enter
will create a 'div' tag</li><li>BR - When 'BR' is configured, pressing enter
or shift + enter will create a 'br' tag</li></ul>`;
  private enterList: DropDownListComponent;
  private popupHeight: string = '200px';
  private enterValue: string = "P";
  private enterPlaceholder: string = "When pressing the enter key";
  private fields: FieldSettingsModel = { text: "text", value: "value" };
  private enterData: { [key: string]: Object }[] = [
    { text: 'Create a new <p>', value: 'P' },
    { text: 'Create a new <div>', value: 'DIV' },
    { text: 'Create a new <br>', value: 'BR' }
  ];
  private enterChange = (): void => {
    if (this.enterList.value === 'P') {
      this.rteObj.enterKey = 'P';
      this.rteObj.value = `<p>In Rich text Editor, the enter key and
shift + enter key actions can be customized using the enterKey and
shiftEnterKey APIs. And the possible values are as follows:</p><ul><li>P -
When 'P' is configured, pressing enter or shift + enter will create a 'p'
tag</li><li>DIV - When 'DIV' is configured, pressing enter or shift + enter
will create a 'div' tag</li><li>BR - When 'BR' is configured, pressing enter
or shift + enter will create a 'br' tag</li></ul>`;
    } else if (this.enterList.value === 'DIV') {
      this.rteObj.enterKey = 'DIV';
      this.rteObj.value = `<div>In Rich text Editor, the enter key and
shift + enter key actions can be customized using the enterKey and
shiftEnterKey APIs. And the possible values are as follows:</div><ul><li>P -
When 'P' is configured, pressing enter or shift + enter will create a 'p'
tag</li><li>DIV - When 'DIV' is configured, pressing enter or shift + enter
will create a 'div' tag</li><li>BR - When 'BR' is configured, pressing enter
or shift + enter will create a 'br' tag</li></ul>`;
    } else if (this.enterList.value === 'BR') {
      this.rteObj.enterKey = 'BR';
      this.rteObj.value = `In Rich text Editor, the enter key and shift
+ enter key actions can be customized using the enterKey and shiftEnterKey
APIs. And the possible values are as follows:<ul><li>P - When 'P' is
configured, pressing enter or shift + enter will create a 'p' tag</li><li>DIV
- When 'DIV' is configured, pressing enter or shift + enter will create a
'div' tag</li><li>BR - When 'BR' is configured, pressing enter or shift +
enter will create a 'br' tag</li></ul>`;
    }
  }
  public render() {
    return (
      <div className='control-pane'>
        <div className='control-section' id="rte">
          <div className='rte-control-section'>
            <table className='api'>
              <tbody>

```

```

        <tr>
            <td>
                <div>
                    <DropDownListComponent id="enterOption"
dataSource={this.enterData} ref={(dropdownlist) => { this.enterList =
dropdownlist }} fields={this.fields} change={this.enterChange.bind(this)}
value={this.enterValue} popupHeight={this.popupHeight}
placeholder={this.enterPlaceholder} floatLabelType="Always" />
                </div>
            </td>
        </tr>
    </tbody>
</table>
<br/>
    <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) =>
{ this.rteObj = richtexteditor }} height={250} value={ this.rteValue }>
        <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]} />
    </RichTextEditorComponent>
</div>
</div>
</div>
);
}
}
export default App;
{% endraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - Enter Key Customization Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import * as React from 'react';
function App() {
    let rteObj;
    let rteValue = `<p>In Rich text Editor, the enter key and shift + enter
key actions can be customized using the enterKey and shiftEnterKey APIs. And
the possible values are as follows:</p><ul><li>P - When 'P' is configured,
pressing enter or shift + enter will create a 'p' tag</li><li>DIV - When
'DIV' is configured, pressing enter or shift + enter will create a 'div'
tag</li><li>BR - When 'BR' is configured, pressing enter or shift + enter
will create a 'br' tag</li></ul>`;
    let enterList;
    let popupHeight = '200px';
    let enterValue = "P";
    let enterPlaceholder = "When pressing the enter key";
    let fields = { text: "text", value: "value" };
    let enterData = [

```

```

    { text: 'Create a new <p>', value: 'P' },
    { text: 'Create a new <div>', value: 'DIV' },
    { text: 'Create a new <br>', value: 'BR' }
  ];
  function enterChange() {
    if (enterList.value === 'P') {
      rteObj.enterKey = 'P';
      rteObj.value = `<p>In Rich text Editor, the enter key and shift +
enter key actions can be customized using the enterKey and shiftEnterKey
APIs. And the possible values are as follows:</p><ul><li>P - When 'P' is
configured, pressing enter or shift + enter will create a 'p' tag</li><li>DIV
- When 'DIV' is configured, pressing enter or shift + enter will create a
'div' tag</li><li>BR - When 'BR' is configured, pressing enter or shift +
enter will create a 'br' tag</li></ul>`;
    }
    else if (enterList.value === 'DIV') {
      rteObj.enterKey = 'DIV';
      rteObj.value = `<div>In Rich text Editor, the enter key and shift
+ enter key actions can be customized using the enterKey and shiftEnterKey
APIs. And the possible values are as follows:</div><ul><li>P - When 'P' is
configured, pressing enter or shift + enter will create a 'p' tag</li><li>DIV
- When 'DIV' is configured, pressing enter or shift + enter will create a
'div' tag</li><li>BR - When 'BR' is configured, pressing enter or shift +
enter will create a 'br' tag</li></ul>`;
    }
    else if (enterList.value === 'BR') {
      rteObj.enterKey = 'BR';
      rteObj.value = `In Rich text Editor, the enter key and shift +
enter key actions can be customized using the enterKey and shiftEnterKey
APIs. And the possible values are as follows:<ul><li>P - When 'P' is
configured, pressing enter or shift + enter will create a 'p' tag</li><li>DIV
- When 'DIV' is configured, pressing enter or shift + enter will create a
'div' tag</li><li>BR - When 'BR' is configured, pressing enter or shift +
enter will create a 'br' tag</li></ul>`;
    }
  }
  return (
    <div className='control-pane'>
      <div className='control-section' id="rte">
        <div className='rte-control-section'>
          <table className='api'>
            <tbody>
              <tr>
                <td>
                  <div>
                    <DropDownListComponent id="enterOption"
dataSource={enterData} ref={(dropdownlist) => { enterList = dropdownlist; }}
fields={fields} change={enterChange.bind(this)} value={enterValue}
popupHeight={popupHeight} placeholder={enterPlaceholder}
floatLabelType="Always"/>
                  </div>
                </td>
              </tr>
            </tbody>
          </table>
          <br />
          <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) =>
{ rteObj = richtexteditor; }} height={250} value={rteValue}>

```

```

        <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]}/>
      </RichTextEditorComponent>
    </div>
  </div>
</div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Enter Key Customization Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { DropDownListComponent, FieldSettingsModel } from '@syncfusion/ej2-
react-dropdowns';
import { createElement } from '@syncfusion/ej2-base';
import * as React from 'react';
function App() {
  let rteObj: RichTextEditorComponent;
  let rteValue: string = `<p>In Rich text Editor, the enter key and shift +
enter key actions can be customized using the enterKey and shiftEnterKey
APIs. And the possible values are as follows:</p><ul><li>P - When 'P' is
configured, pressing enter or shift + enter will create a 'p' tag</li><li>DIV
- When 'DIV' is configured, pressing enter or shift + enter will create a
'div' tag</li><li>BR - When 'BR' is configured, pressing enter or shift +
enter will create a 'br' tag</li></ul>`;
  let enterList: DropDownListComponent;
  let popupHeight: string = '200px';
  let enterValue: string = 'P';
  let enterPlaceholder: string = "When pressing the enter key";
  let fields: FieldSettingsModel = { text: "text", value: "value" };
  let enterData: { [key: string]: Object }[] = [
    { text: 'Create a new <p>', value: 'P' },
    { text: 'Create a new <div>', value: 'DIV' },
    { text: 'Create a new <br>', value: 'BR' }
  ];
  function enterChange (): void {
    if (enterList.value === 'P') {
      rteObj.enterKey = 'P';
      rteObj.value = `<p>In Rich text Editor, the enter key and shift +
enter key actions can be customized using the enterKey and shiftEnterKey
APIs. And the possible values are as follows:</p><ul><li>P - When 'P' is
configured, pressing enter or shift + enter will create a 'p' tag</li><li>DIV
- When 'DIV' is configured, pressing enter or shift + enter will create a
'div' tag</li><li>BR - When 'BR' is configured, pressing enter or shift +
enter will create a 'br' tag</li></ul>`;
    } else if (enterList.value === 'DIV') {
      rteObj.enterKey = 'DIV';
      rteObj.value = `<div>In Rich text Editor, the enter key and shift
+ enter key actions can be customized using the enterKey and shiftEnterKey

```

```

APIs. And the possible values are as follows:</div><ul><li>P - When 'P' is
configured, pressing enter or shift + enter will create a 'p' tag</li><li>DIV
- When 'DIV' is configured, pressing enter or shift + enter will create a
'div' tag</li><li>BR - When 'BR' is configured, pressing enter or shift +
enter will create a 'br' tag</li></ul>`;
    } else if (enterList.value === 'BR') {
        rteObj.enterKey = 'BR';
        rteObj.value = `In Rich text Editor, the enter key and shift +
enter key actions can be customized using the enterKey and shiftEnterKey
APIs. And the possible values are as follows:<ul><li>P - When 'P' is
configured, pressing enter or shift + enter will create a 'p' tag</li><li>DIV
- When 'DIV' is configured, pressing enter or shift + enter will create a
'div' tag</li><li>BR - When 'BR' is configured, pressing enter or shift +
enter will create a 'br' tag</li></ul>`;
    }
}
return (
    <div className='control-pane'>
        <div className='control-section' id="rte">
            <div className='rte-control-section'>
                <table className='api'>
                    <tbody>
                        <tr>
                            <td>
                                <div>
                                    <DropDownListComponent id="enterOption"
dataSource={enterData} ref={(dropdownlist) => { enterList = dropdownlist }}
fields={fields} change={enterChange.bind(this)} value={enterValue}
popupHeight={popupHeight} placeholder={enterPlaceholder}
floatLabelType="Always" />
                                </div>
                            </td>
                        </tr>
                    </tbody>
                </table>
                <br/>
                <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) =>
{ rteObj = richtexteditor }} height={250} value={ rteValue }>
                    <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]} />
                </RichTextEditorComponent>
            </div>
        </div>
    </div>
);
}
export default App;
{% endraw %}

```

Shift-Enter key customization

By default, the `
` tag will be created while pressing the shift + enter key. The shift + enter key can be customized by using the [shiftEnterKey](#) property where the possible tags that can be used to customize are `
`, `<p>`, `<div>`.

When the shift + enter key is customized with any of the possible values, pressing the shift + enter key in the editor will create a new tag that is configured. Also, when the shift + enter key is configured the default value of the Rich Text Editor will change respectively with the configured values.

[Class-component]

APP.JSX

```
{% raw %}
/**
 * Rich Text Editor - Shift Enter Key Customization Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  rteValue = `

In Rich text Editor, the enter key and shift + enter key
actions can be customized using the enterKey and shiftEnterKey APIs. And the
possible values are as follows:</p><ul><li>P - When 'P' is configured,
pressing enter or shift + enter will create a 'p' tag</li><li>DIV - When
'DIV' is configured, pressing enter or shift + enter will create a 'div'
tag</li><li>BR - When 'BR' is configured, pressing enter or shift + enter
will create a 'br' tag</li></ul>`;
  enterList;
  shiftEnterList;
  popupHeight = '200px';
  shiftEnterValue = "BR";
  shiftEnterPlaceholder = "When pressing the shift + enter key";
  fields = { text: "text", value: "value" };
  shiftEnterData = [
    { text: 'Create a new <br>', value: 'BR' },
    { text: 'Create a new <div>', value: 'DIV' },
    { text: 'Create a new <p>', value: 'P' }
  ];
  shiftEnterChange = () => {
    if (this.shiftEnterList.value === 'BR') {
      this.rteObj.shiftEnterKey = 'BR';
    }
    else if (this.shiftEnterList.value === 'DIV') {
      this.rteObj.shiftEnterKey = 'DIV';
    }
    else if (this.shiftEnterList.value === 'P') {
      this.rteObj.shiftEnterKey = 'P';
    }
  };
  render() {
    return (<div className='control-pane'>
      <div className='control-section' id="rte">
        <div className='rte-control-section'>
          <table className='api'>
            <tbody>
              <tr>
                <td>
                  <div>


```

```

                                <DropDownListComponent id="shiftEnterOption"
dataSource={this.shiftEnterData} ref={(dropdownlist) => { this.shiftEnterList
= dropdownlist; }} fields={this.fields}
change={this.shiftEnterChange.bind(this)} value={this.shiftEnterValue}
popupHeight={this.popupHeight} placeholder={this.shiftEnterPlaceholder}
floatLabelType="Always"/>
                                </div>
                                </td>
                                </tr>
                                </tbody>
                                </table>
                                <br />
                                <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) =>
{ this.rteObj = richtexteditor; }} height={250} value={this.rteValue}>
                                <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]}/>
                                </RichTextEditorComponent>
                                </div>
                                </div>
                                </div>);
        }
    }
    export default App;
    {% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Shift Enter Key Customization Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { DropDownListComponent, FieldSettingsModel } from '@syncfusion/ej2-
react-dropdowns';
import { createElement } from '@syncfusion/ej2-base';
import * as React from 'react';
class App extends React.Component<{}, {}> {
    private rteObj: RichTextEditorComponent;
    private rteValue: string = `<p>In Rich text Editor, the enter key and shift
+ enter key actions can be customized using the enterKey and shiftEnterKey
APIs. And the possible values are as follows:</p><ul><li>P - When 'P' is
configured, pressing enter or shift + enter will create a 'p' tag</li><li>DIV
- When 'DIV' is configured, pressing enter or shift + enter will create a
'div' tag</li><li>BR - When 'BR' is configured, pressing enter or shift +
enter will create a 'br' tag</li></ul>`;
    private enterList: DropDownListComponent;
    private shiftEnterList: DropDownListComponent;
    private popupHeight: string = '200px';
    private shiftEnterValue: string = "BR";
    private shiftEnterPlaceholder: string = "When pressing the shift + enter
key";
    private fields: FieldSettingsModel = { text: "text", value: "value" };
    private shiftEnterData: { [key: string]: Object }[] = [
        { text: 'Create a new <br>', value: 'BR' },

```

```

    { text: 'Create a new <div>', value: 'DIV' },
    { text: 'Create a new <p>', value: 'P' }
  ];
  private shiftEnterChange = (): void => {
    if (this.shiftEnterList.value === 'BR') {
      this.rteObj.shiftEnterKey = 'BR';
    } else if (this.shiftEnterList.value === 'DIV') {
      this.rteObj.shiftEnterKey = 'DIV';
    } else if (this.shiftEnterList.value === 'P') {
      this.rteObj.shiftEnterKey = 'P';
    }
  }
  public render() {
    return (
      <div className='control-pane'>
        <div className='control-section' id="rte">
          <div className='rte-control-section'>
            <table className='api'>
              <tbody>
                <tr>
                  <td>
                    <div>
                      <DropDownListComponent id="shiftEnterOption"
dataSource={this.shiftEnterData} ref={(dropdownlist) => { this.shiftEnterList
= dropdownlist }} fields={this.fields}
change={this.shiftEnterChange.bind(this)} value={this.shiftEnterValue}
popupHeight={this.popupHeight} placeholder={this.shiftEnterPlaceholder}
floatLabelType="Always" />
                    </div>
                  </td>
                </tr>
              </tbody>
            </table>
            <br/>
            <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) =>
{ this.rteObj = richtexteditor }} height={250} value={ this.rteValue }>
              <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]} />
            </RichTextEditorComponent>
          </div>
        </div>
      </div>
    );
  }
}
export default App;
{% endraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - Shift Enter Key Customization Sample
 */

```

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import * as React from 'react';
function App() {
    let rteObj;
    let rteValue = `

In Rich text Editor, the enter key and shift + enter
key actions can be customized using the enterKey and shiftEnterKey APIs. And
the possible values are as follows:</p><ul><li>P - When 'P' is configured,
pressing enter or shift + enter will create a 'p' tag</li><li>DIV - When
'DIV' is configured, pressing enter or shift + enter will create a 'div'
tag</li><li>BR - When 'BR' is configured, pressing enter or shift + enter
will create a 'br' tag</li></ul>`;
    let enterList;
    let shiftEnterList;
    let popupHeight = '200px';
    let shiftEnterValue = "BR";
    let shiftEnterPlaceholder = "When pressing the shift + enter key";
    let fields = { text: "text", value: "value" };
    let shiftEnterData = [
        { text: 'Create a new <br>', value: 'BR' },
        { text: 'Create a new <div>', value: 'DIV' },
        { text: 'Create a new <p>', value: 'P' }
    ];
    function shiftEnterChange() {
        if (shiftEnterList.value === 'BR') {
            rteObj.shiftEnterKey = 'BR';
        }
        else if (shiftEnterList.value === 'DIV') {
            rteObj.shiftEnterKey = 'DIV';
        }
        else if (shiftEnterList.value === 'P') {
            rteObj.shiftEnterKey = 'P';
        }
    }
    return (<div className='control-pane'>
        <div className='control-section' id="rte">
            <div className='rte-control-section'>
                <table className='api'>
                    <tbody>
                        <tr>
                            <td>
                                <div>
                                    <DropDownListComponent id="shiftEnterOption"
dataSource={shiftEnterData} ref={(dropdownlist) => { shiftEnterList =
dropdownlist; }} fields={fields} change={shiftEnterChange.bind(this)}
value={shiftEnterValue} popupHeight={popupHeight}
placeholder={shiftEnterPlaceholder} floatLabelType="Always"/>
                                </div>
                            </td>
                        </tr>
                    </tbody>
                </table>
                <br />
                <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) => {
rteObj = richtexteditor; }} height={250} value={rteValue}>


```

```

        <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]}/>
    </RichTextEditorComponent>
</div>
</div>
</div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Shift Enter Key Customization Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import { DropDownListComponent, FieldSettingsModel } from '@syncfusion/ej2-
react-dropdowns';
import { createElement } from '@syncfusion/ej2-base';
import * as React from 'react';
function App() {
    let rteObj: RichTextEditorComponent;
    let rteValue: string = `<p>In Rich text Editor, the enter key and shift +
enter key actions can be customized using the enterKey and shiftEnterKey
APIs. And the possible values are as follows:</p><ul><li>P - When 'P' is
configured, pressing enter or shift + enter will create a 'p' tag</li><li>DIV
- When 'DIV' is configured, pressing enter or shift + enter will create a
'div' tag</li><li>BR - When 'BR' is configured, pressing enter or shift +
enter will create a 'br' tag</li></ul>`;
    let enterList: DropDownListComponent;
    let shiftEnterList: DropDownListComponent;
    let popupHeight: string = '200px';
    let shiftEnterValue: string = "BR";
    let shiftEnterPlaceholder: string = "When pressing the shift + enter key";
    let fields: FieldSettingsModel = { text: "text", value: "value" };
    let shiftEnterData: { [key: string]: Object }[] = [
        { text: 'Create a new <br>', value: 'BR' },
        { text: 'Create a new <div>', value: 'DIV' },
        { text: 'Create a new <p>', value: 'P' }
    ];
};
function shiftEnterChange(): void {
    if (shiftEnterList.value === 'BR') {
        rteObj.shiftEnterKey = 'BR';
    } else if (shiftEnterList.value === 'DIV') {
        rteObj.shiftEnterKey = 'DIV';
    } else if (shiftEnterList.value === 'P') {
        rteObj.shiftEnterKey = 'P';
    }
}
return (
    <div className='control-pane'>
        <div className='control-section' id="rte">
            <div className='rte-control-section'>

```

```

        <table className='api'>
            <tbody>
                <tr>
                    <td>
                        <div>
                            <DropDownListComponent id="shiftEnterOption"
dataSource={shiftEnterData} ref={(dropdownlist) => { shiftEnterList =
dropdownlist }} fields={fields} change={shiftEnterChange.bind(this)}
value={shiftEnterValue} popupHeight={popupHeight}
placeholder={shiftEnterPlaceholder} floatLabelType="Always" />
                        </div>
                    </td>
                </tr>
            </tbody>
        </table>
        <br/>
        <RichTextEditorComponent id="defaultRTE" ref={(richtexteditor) => {
rteObj = richtexteditor }} height={250} value={ rteValue }>
            <Inject services={[HtmlEditor, Toolbar, Image, Link,
QuickToolbar]} />
        </RichTextEditorComponent>
    </div>
</div>
</div>
);
}
export default App;
{% endraw %}

```

Iframe in React Rich text editor component

When the `iframeSettings` option is enabled, the Rich Text Editor creates the `iframe` element as the content area on control initialization; it is used to display and editing the content. In content area, the editor displays only the body tag of a `<iframe>` document.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - IFrame sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
    iframeSettings = {
        enable: true
    };
    render() {
        return (<RichTextEditorComponent height={450}
iframeSettings={this.iframeSettings}>
            <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
            Users can format their content using standard toolbar commands.</p>

```

```

    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
  </RichTextEditorComponent>;
}
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - IFrame sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{},{}> {
  private iframeSettings: object = {
    enable: true
  }
  public render() {

```

```

    return (
      <RichTextEditorComponent height={450}
        iframeSettings={this.iframeSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
        you get") editor that provides the best user experience to create and update
        the content.
          Users can format their content using standard toolbar commands.</p>
          <p><b>Key features:</b></p>
          <ul>
            <li>
              <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
              <p>Capable of handling markdown editing.</p>
            </li>
            <li>
              <p>Contains a modular library to load the necessary functionality
on demand.</p>
            </li>
            <li>
              <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
              <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
              <p>Supports third-party library integration.</p>
            </li>
            <li>
              <p>Allows preview of modified content before saving it.</p>
            </li>
            <li>
              <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
            </li>
            <li>
              <p>Contains undo/redo manager.</p>
            </li>
            <li>
              <p>Creates bulleted and numbered lists.</p>
            </li>
          </ul>
          <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
        />
      </RichTextEditorComponent>
    );
  }
}
export default App;

```

[Functional-component]

APP.JSX

/**


```

* Rich Text Editor - IFrame sample
*/
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let iframeSettings = {
    enable: true
  };
  return (<RichTextEditorComponent height={450}
iframeSettings={iframeSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>
    </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - IFrame sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let iframeSettings: object = {
    enable: true
  }
  return (
    <RichTextEditorComponent height={450} iframeSettings={iframeSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}

```

```
}
export default App;
```

Iframe attributes

The editor allows you to pass an additional attribute to body tag of a `<iframe>` element using attributes fields of the [iframeSettings](#) property. This property contains name/value pairs in string format. It is used to override the default appearance of the content area.

[Class-component]

APP.JSX

```
/**
 * Rich Text Editor - IFrame Attributes sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  iframeSettings = {
    attributes: {
      readonly: 'readonly'
    },
    enable: true,
  };
  render() {
    return (<RichTextEditorComponent height={450}
iframeSettings={this.iframeSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
    </ul>
  );
}
```

```

        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
  </RichTextEditorComponent>;
}
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - IFrame Attributes sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{},{}> {
  private iframeSettings: object = {
    attributes: {
      readonly: 'readonly'
    },
    enable: true,
  }
  public render() {
    return (
      <RichTextEditorComponent height={450}
iframeSettings={this.iframeSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>

```

```

        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
  </RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - IFrame Attributes sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let iframeSettings = {
    attributes: {
      readonly: 'readonly'
    },
    enable: true,
  };
  return (<RichTextEditorComponent height={450}
iframeSettings={iframeSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>

```

```

        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
          </li>
          <li>
            <p>Contains undo/redo manager.</p>
          </li>
          <li>
            <p>Creates bulleted and numbered lists.</p>
          </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
      </RichTextEditorComponent>;
    }
    export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - IFrame Attributes sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let iframeSettings: object = {
    attributes: {
      readonly: 'readonly'
    },
    enable: true,
  }
  return (

```

```

<RichTextEditorComponent height={450} iframeSettings={iframeSettings}>
  <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
  get") editor that provides the best user experience to create and update the
  content.
    Users can format their content using standard toolbar commands.</p>
  <p><b>Key features:</b></p>
  <ul>
    <li>
      <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
    </li>
    <li>
      <p>Capable of handling markdown editing.</p>
    </li>
    <li>
      <p>Contains a modular library to load the necessary functionality
      on demand.</p>
    </li>
    <li>
      <p>Provides a fully customizable toolbar.</p>
    </li>
    <li>
      <p>Provides HTML view to edit the source directly for
      developers.</p>
    </li>
    <li>
      <p>Supports third-party library integration.</p>
    </li>
    <li>
      <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
      <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
    </li>
    <li>
      <p>Contains undo/redo manager.</p>
    </li>
    <li>
      <p>Creates bulleted and numbered lists.</p>
    </li>
  </ul>
  <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
export default App;

```

Adding external CSS/Script file

The editor offers you to add external CSS file to style the `<iframe>` element. Easily change the appearance of editor's content using an external CSS file using [styles](#) field in the `iframeSettings` property.

Likewise, add the external script file to the `<iframe>` element using the [scripts](#) field of `iframeSettings` to provide the additional functionalities to the Rich Text Editor.

[Class-component]

APP.JSX

```

/**
 * Rich Text Editor - IFrame Resources sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  iframeSettings = {
    enable: true,
    resources: {
      scripts: [],
      styles: ['myStyle.css'] // Styles
    }
  };
  render() {
    return (<RichTextEditorComponent height={450}
iframeSettings={this.iframeSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b> <label> --> Styles injected from iframe
settings</label></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>

```



```

        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
    </RichTextEditorComponent>);
  }
}
export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - IFrame Resources sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private iframeSettings: object = {
    enable: true,
    resources: {
      scripts: [], // Scripts
      styles: ['myStyle.css'] // Styles
    }
  }
  public render() {
    return (
      <RichTextEditorComponent height={450}
iframeSettings={this.iframeSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b> <label> --> Styles injected from iframe
settings</label></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>

```

```

        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - IFrame Resources sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let iframeSettings = {
    enable: true,
    resources: {
      scripts: [],
      styles: ['myStyle.css'] // Styles
    }
  };
  return (<RichTextEditorComponent height={450}
iframeSettings={iframeSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b> <label> --> Styles injected from iframe
settings</label></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
    </ul>

```

```

        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
    </RichTextEditorComponent>);
  }
  export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - IFrame Resources sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let iframeSettings: object = {
    enable: true,
    resources: {
      scripts: [], // Scripts
      styles: ['myStyle.css'] // Styles
    }
  }
  return (
    <RichTextEditorComponent height={450} iframeSettings={iframeSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
    </RichTextEditorComponent>
  );
}

```

```

    <p><b>Key features:</b> <label> --> Styles injected from iframe
    settings</label></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
  </RichTextEditorComponent>
);
}
export default App;

```

Third party integration in React Rich text editor component

The Rich Text Editor can be integrated with third-party to suite the application scenario.

CodeMirror Integration

Rich Text Editor comes with a basic HTML source editor through the view-source property. CodeMirror plugin can be used to highlight the syntax of HTML. CodeMirror plugin for Rich Text Editor makes editing of HTML source code with a pleasant experience.

Import necessary CSS and JS files of CodeMirror to the HTML page.

Required JS files of code mirror.

```
` javascript
```

```
<script src="scripts/CodeMirror/codemirror.js" type="text/javascript"></script>
<script src="scripts/CodeMirror/javascript.js" type="text/javascript"></script>
<script src="scripts/CodeMirror/css.js" type="text/javascript"></script>
<script src="scripts/CodeMirror/htmlmixed.js" type="text/javascript"></script>
```

Required CSS file of code mirror.

```
` javascript
<link href="scripts/CodeMirror/codemirror.min.css" rel="stylesheet" />
```

Add a custom icon for HTML source editor in the toolbar of Rich Text Editor using the template option of ToolbarSettings, define the code mirror plugins, and then pass the Rich Text Editor content as argument in the [actionComplete](#) event.

[Class-component]

APP.JSX

```
{% raw %}
/**
 * Rich Text Editor - Code Mirror sample
 */
import { createElement } from '@syncfusion/ej2-base';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import CodeMirror from 'codemirror';
import 'codemirror/mode/css/css.js';
import 'codemirror/mode/htmlmixed/htmlmixed.js';
import 'codemirror/mode/javascript/javascript';
import * as React from 'react';
class App extends React.Component {
  myCodeMirror;
  textArea;
  rteObj;
  toolbarSettings = {
    items: ['SourceCode']
  };
  value = `
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
    get") editor that provides the best user experience to create and update the
    content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
```

```

        <p>Contains a modular library to load the necessary functionality
on demand.</p>
    </li>
    <li>
        <p>Provides a fully customizable toolbar.</p>
    </li>
    <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
    </li>
    <li>
        <p>Supports third-party library integration.</p>
    </li>
    <li>
        <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
    </li>
    <li>
        <p>Contains undo/redo manager.</p>
    </li>
    <li>
        <p>Creates bulleted and numbered lists.</p>
    </li>
</ul>
`
;
mirrorConversion(e) {
    const id = this.rteObj.getID() + 'mirror-view';
    let mirrorView = this.rteObj.element.querySelector('#' + id);
    const charCount = this.rteObj.element.querySelector('.e-rte-
character-count');
    if (e.targetItem === 'Preview') {
        this.textArea.style.display = 'block';
        mirrorView.style.display = 'none';
        this.textArea.innerHTML = this.myCodeMirror.getValue();
        charCount.style.display = 'block';
    }
    else {
        if (!mirrorView) {
            mirrorView = createElement('div', { className: 'e-content'
});
            mirrorView.id = id;
            this.textArea.parentNode.appendChild(mirrorView);
        }
        else {
            mirrorView.innerHTML = '';
        }
        this.textArea.style.display = 'none';
        mirrorView.style.display = 'block';
        this.renderCodeMirror(mirrorView, this.rteObj.value);
        charCount.style.display = 'none';
    }
}
renderCodeMirror(mirrorView, content) {
    this.myCodeMirror = CodeMirror(mirrorView, {

```

```

        lineNumbers: true,
        lineWrapping: true,
        mode: 'text/html',
        value: content
    });
}
actionComplete(e) {
    if (e.targetItem && (e.targetItem === 'SourceCode' || e.targetItem === 'Preview')) {
        this.rteObj.sourceCodeModule.getPanel().style.display = 'none';
        this.mirrorConversion(e);
    }
    else {
        setTimeout(() => {
            this.rteObj.toolbarModule.refreshToolbarOverflow();
        }, 400);
    }
}
created() {
    this.textArea = this.rteObj.contentModule.getEditPanel();
}
render() {
    return (<RichTextEditorComponent ref={ (richtexteditor) => {
this.rteObj = richtexteditor; }} height={450}
toolbarSettings={this.toolbarSettings} showCharCount={true}
actionComplete={this.actionComplete = this.actionComplete.bind(this)}
created={this.created = this.created.bind(this)}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
                <p>Capable of handling markdown editing.</p>
            </li>
            <li>
                <p>Contains a modular library to load the necessary functionality
on demand.</p>
            </li>
            <li>
                <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
                <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
                <p>Supports third-party library integration.</p>
            </li>
            <li>
                <p>Allows preview of modified content before saving it.</p>
            </li>
        </ul>
    </RichTextEditorComponent>
    );
}

```

```

        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
    </li>
    <li>
        <p>Contains undo/redo manager.</p>
    </li>
    <li>
        <p>Creates bulleted and numbered lists.</p>
    </li>
</ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]}/>
    </RichTextEditorComponent>);
    }
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Code Mirror sample
 */
import { createElement } from '@syncfusion/ej2-base';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import CodeMirror from 'codemirror';
import 'codemirror/mode/css/css.js';
import 'codemirror/mode/htmlmixed/htmlmixed.js';
import 'codemirror/mode/javascript/javascript';
import * as React from 'react';
class App extends React.Component<{},{}> {
    public myCodeMirror: any;
    public textArea: HTMLInputElement;
    public rteObj: RichTextEditorComponent;
    public toolbarSettings: object = {
        items: ['SourceCode']
    }
    public value: string = `
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.

    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
    <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
    </li>
    <li>
        <p>Capable of handling markdown editing.</p>
    </li>
    <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>

```



```

        </li>
        <li>
            <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
            <p>Supports third-party library integration.</p>
        </li>
        <li>
            <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    `;
    public mirrorConversion(e: any): void {
        const id: string = this.rteObj.getID() + 'mirror-view';
        let mirrorView: HTMLElement = this.rteObj.element.querySelector('#' + id)
as HTMLElement;
        const charCount: HTMLElement = this.rteObj.element.querySelector('.e-rte-
character-count') as HTMLElement;
        if (e.targetItem === 'Preview') {
            this.textArea.style.display = 'block';
            mirrorView.style.display = 'none';
            this.textArea.innerHTML = this.myCodeMirror.getValue();
            charCount.style.display = 'block';
        } else {
            if (!mirrorView) {
                mirrorView = createElement('div', { className: 'e-content' });
                mirrorView.id = id;
                (this.textArea as any).parentNode.appendChild(mirrorView);
            } else {
                mirrorView.innerHTML = '';
            }
            this.textArea.style.display = 'none';
            mirrorView.style.display = 'block';
            this.renderCodeMirror(mirrorView, this.rteObj.value);
            charCount.style.display = 'none';
        }
    }
    public renderCodeMirror(mirrorView: HTMLElement, content: string): void {
        this.myCodeMirror = CodeMirror(mirrorView, {
            lineNumbers: true,
            lineWrapping: true,
            mode: 'text/html',
            value: content
        });
    }
}

```

```

    });
  }
  public actionComplete(e: any) {
    if (e.targetItem && (e.targetItem === 'SourceCode' || e.targetItem ===
'Preview')) {
      (this.rteObj as any).sourceCodeModule.getPanel().style.display =
'none';
      this.mirrorConversion(e);
    } else {
      setTimeout(() => {
        this.rteObj.toolbarModule.refreshToolbarOverflow();
      }, 400);
    }
  }
  public created(): void{
    this.textArea = (this.rteObj as any).contentModule.getEditPanel() as
HTMLInputElement;
  }
  public render() {
    return (
      <RichTextEditorComponent ref={(richtexteditor) => { this.rteObj =
richtexteditor! }} height={450} toolbarSettings={this.toolbarSettings}
showCharCount={true} actionComplete={this.actionComplete} =
this.actionComplete.bind(this)} created={this.created} =
this.created.bind(this)}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>

```

```

        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]} />
    </RichTextEditorComponent>
  );
}
}
export default App;
{% endraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - Code Mirror sample
 */
import { createElement } from '@syncfusion/ej2-base';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import CodeMirror from 'codemirror';
import 'codemirror/mode/css/css.js';
import 'codemirror/mode/htmlmixed/htmlmixed.js';
import 'codemirror/mode/javascript/javascript.js';
import * as React from 'react';
function App() {
  let myCodeMirror;
  let textArea;
  let rteObj;
  let toolbarSettings = {
    items: ['SourceCode']
  };
  let value = `
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.

    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>

```

```

        <p>Contains a modular library to load the necessary functionality
on demand.</p>
    </li>
    <li>
        <p>Provides a fully customizable toolbar.</p>
    </li>
    <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
    </li>
    <li>
        <p>Supports third-party library integration.</p>
    </li>
    <li>
        <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
    </li>
    <li>
        <p>Contains undo/redo manager.</p>
    </li>
    <li>
        <p>Creates bulleted and numbered lists.</p>
    </li>
</ul>
`
;
function mirrorConversion(e) {
    const id = rteObj.getID() + 'mirror-view';
    let mirrorView = rteObj.element.querySelector('#' + id);
    const charCount = rteObj.element.querySelector('.e-rte-character-
count');
    if (e.targetItem === 'Preview') {
        textArea.style.display = 'block';
        mirrorView.style.display = 'none';
        textArea.innerHTML = myCodeMirror.getValue();
        charCount.style.display = 'block';
    }
    else {
        if (!mirrorView) {
            mirrorView = createElement('div', { className: 'e-content'
});
            mirrorView.id = id;
            textArea.parentNode.appendChild(mirrorView);
        }
        else {
            mirrorView.innerHTML = '';
        }
        textArea.style.display = 'none';
        mirrorView.style.display = 'block';
        renderCodeMirror(mirrorView, rteObj.value);
        charCount.style.display = 'none';
    }
}
function renderCodeMirror(mirrorView, content) {
    myCodeMirror = CodeMirror(mirrorView, {

```

```

        lineNumbers: true,
        lineWrapping: true,
        mode: 'text/html',
        value: content
    });
}
function actionComplete(e) {
    if (e.targetItem && (e.targetItem === 'SourceCode' || e.targetItem
=== 'Preview')) {
        rteObj.sourceCodeModule.getPanel().style.display = 'none';
        mirrorConversion(e);
    }
    else {
        setTimeout(() => {
            rteObj.toolbarModule.refreshToolbarOverflow();
        }, 400);
    }
}
function created() {
    textArea = rteObj.contentModule.getEditPanel();
}
return (<RichTextEditorComponent ref={ (richtexteditor) => { rteObj =
richtexteditor; }} height={450} toolbarSettings={toolbarSettings}
showCharCount={true} actionComplete={actionComplete.bind(this)}
created={created.bind(this)}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
        <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
            <p>Capable of handling markdown editing.</p>
        </li>
        <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
            <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
            <p>Supports third-party library integration.</p>
        </li>
        <li>
            <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
    </ul>

```

```

        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]}/>
    </RichTextEditorComponent>;
  }
  export default App;
  {% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Code Mirror sample
 */
import { createElement } from '@syncfusion/ej2-base';
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import CodeMirror from 'codemirror';
import 'codemirror/mode/css/css.js';
import 'codemirror/mode/htmlmixed/htmlmixed.js';
import 'codemirror/mode/javascript/javascript';
import * as React from 'react';
function App() {
  let myCodeMirror: any;
  let textArea: HTMLInputElement;
  let rteObj: RichTextEditorComponent;
  let toolbarSettings: object = {
    items: ['SourceCode']
  }
  let value: string = `
  <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.

  Users can format their content using standard toolbar commands.</p>
  <p><b>Key features:</b></p>
  <ul>
    <li>
      <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
    </li>
    <li>
      <p>Capable of handling markdown editing.</p>
    </li>
    <li>
      <p>Contains a modular library to load the necessary functionality
on demand.</p>
    </li>
    <li>
      <p>Provides a fully customizable toolbar.</p>

```

```

        </li>
        <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
            <p>Supports third-party library integration.</p>
        </li>
        <li>
            <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
    `;
    function mirrorConversion(e: any): void {
        const id: string = rteObj.getID() + 'mirror-view';
        let mirrorView: HTMLElement = rteObj.element.querySelector('#' + id) as
HTMLElement;
        const charCount: HTMLElement = rteObj.element.querySelector('.e-rte-
character-count') as HTMLElement;
        if (e.targetItem === 'Preview') {
            textArea.style.display = 'block';
            mirrorView.style.display = 'none';
            textArea.innerHTML = myCodeMirror.getValue();
            charCount.style.display = 'block';
        } else {
            if (!mirrorView) {
                mirrorView = createElement('div', { className: 'e-content' });
                mirrorView.id = id;
                (textArea as any).parentNode.appendChild(mirrorView);
            } else {
                mirrorView.innerHTML = '';
            }
            textArea.style.display = 'none';
            mirrorView.style.display = 'block';
            renderCodeMirror(mirrorView, rteObj.value);
            charCount.style.display = 'none';
        }
    }
    function renderCodeMirror(mirrorView: HTMLElement, content: string): void {
        myCodeMirror = CodeMirror(mirrorView, {
            lineNumbers: true,
            lineWrapping: true,
            mode: 'text/html',
            value: content
        });
    }
    function actionComplete(e: any) {

```

```

    if (e.targetItem && (e.targetItem === 'SourceCode' || e.targetItem ===
'Preview')) {
      (rteObj as any).sourceCodeModule.getPanel().style.display = 'none';
      mirrorConversion(e);
    } else {
      setTimeout(() => {
        rteObj.toolbarModule.refreshToolbarOverflow();
      }, 400);
    }
  }
}
function created(): void{
  textArea = (rteObj as any).contentModule.getEditPanel() as HTMLElement;
}
return (
  <RichTextEditorComponent ref={(richtexteditor) => { rteObj =
richtexteditor! }} height={450} toolbarSettings={toolbarSettings}
showCharCount={true} actionComplete={actionComplete.bind(this)}
created={created.bind(this)}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
  </RichTextEditorComponent>
)

```



```

        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar,
Count]} />
      </RichTextEditorComponent>
    );
  }
  export default App;
  {% endraw %}

```

Embed.ly Integration

Rich Text Editor easily integrate with embed.ly which is probably the best service when it comes to embed the rich content such as Twitter, Facebook, Instagram, and lots of other publishing platform embeds.

` javascript

```
<script src="https://cdn.embedly.com/widgets/platform.js" charset="UTF-8"></script>
```

`

In the following sample, the Embed.ly class `embedly-card` has been added to a `a` tag in the [actionComplete](#) event.

Exec command in React Rich text editor component

In Rich Text Editor, `execCommand` used to perform commands for the modification of content in editable area.

The `execCommand` will perform the following commands.

Commands	Description	Code snippets
<code>bold</code>	Bold the selected content in the Rich Text Editor.	<code>rteObj.executeCommand('bold');</code>
<code>italic</code>	The selected text will be italics.	<code>rteObj.executeCommand('italic');</code>
<code>underline</code>	Underline the selected text in the Rich Text Editor.	<code>rteObj.executeCommand('underline');</code>
<code>strikeThrough</code>	Apply single line strike through formatting for the selected text.	<code>rteObj.executeCommand('strikeThrough');</code>
<code>superscript</code>	Makes the selected text as superscript (higher).	<code>rteObj.executeCommand('superscript');</code>
<code>subscript</code>	Makes the selected text as subscript (lower).	<code>rteObj.executeCommand('subscript');</code>
<code>uppercase</code>	Change the case of selected text to upper in the content.	<code>rteObj.executeCommand('uppercase');</code>
<code>lowercase</code>	Change the case of selected text to lower in the content.	<code>rteObj.executeCommand('lowercase');</code>
<code>fontColor</code>	Apply the specified font color for the selected text.	<code>rteObj.executeCommand('fontColor', 'yellow');</code>

| **fontName** | Apply the specified font name for the selected text.
`rteObj.executeCommand('fontName', 'Arial');`|

| **fontSize** | Apply the specified font size for the selected text. | `rteObj.executeCommand('fontSize', '10pt');`|

| **formatBlock** | Apply the specified format styles for the selected text.
`rteObj.executeCommand('formatBlock', 'H1');`|

| **backColor** | Apply the specified background color the selected text. |
`rteObj.executeCommand('backColor', 'red');`|

| **justifyCenter** | Align the content with center margin. | `rteObj.executeCommand('justifyCenter');`|

| **justifyFull** | Align the content with justify margin. | `rteObj.executeCommand('justifyFull');`|

| **justifyLeft** | Align the content with left margin. | `rteObj.executeCommand('justifyLeft');`|

| **justifyRight** | Align the content with right margin. | `rteObj.executeCommand('justifyLeft');`|

| **undo** | Allows to undo the actions. | `rteObj.executeCommand('undo');`|

| **createLink** | Creates a hyperlink to a text or image to a specific location in the content. |
`rteObj.executeCommand('createLink',{ text: 'Links', url: 'http://', title : 'Link' });` |

| **indent** | Allows to increase the indent level of the content. | `rteObj.executeCommand('indent');`|

| **insertHTML** | Insert the html content to the current cursor position. |
`rteObj.executeCommand('insertHTML', 'inserted an html');`|

| **insertOrderedList** | Create a new list item(numbered). |
`rteObj.executeCommand('insertOrderedList');`|

| **insertUnorderedList** | Create a new list item(bulleted).
`rteObj.executeCommand('insertUnorderedList');`|

| **outdent** | Allows to decrease the indent level of the content. |
`rteObj.executeCommand('outdent');`|

| **redo** | Allows to redo the actions | `rteObj.executeCommand('redo');`|

| **removeFormat** | remove all formatting styles (such as bold, italic, underline, color, superscript, subscript, and more) from currently selected text. | `rteObj.executeCommand('removeFormat');`|

| **insertText** | Insert text to the current cursor position. | `rteObj.executeCommand('insertText', 'inserted a text');` |

| **insertImage** | Insert an image to the current cursor position. |
`rteObj.executeCommand('insertImage', { url: 'https://ej2.syncfusion.com/javascript/demos/src/rich-text-editor/images/RTEImage-Feather.png', cssClass: 'rte-img' });`

| **copyFormatPainter** | Copy the format of selected text and apply it to another text. |
`rteObj.executeCommand('copyFormatPainter', formatPainterSettings);`|

| **applyFormatPainter** | Apply the copied format to the selected text. |
`rteObj.executeCommand('applyFormatPainter');`|

```
| escapeFormatPainter | Remove the previously copied format and disable the sticky mode |
rteObj.executeCommand('escapeFormatPainter');|
```

Note: The 'ExecuteCommand' public method is not supported in Syncfusion Markdown Editor

[Style in React Rich text editor component](#)

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

[Customizing the Rich Text Editor's content](#)

Use the following CSS to customize the default Rich Text Editor's content properties like font-family, font-size and color.

```
`css
/ To change font family and font size /
.e-richtexteditor .e-rte-content .e-content,
.e-richtexteditor .e-source-content .e-content {
font-size: 20px;
font-family: Segoe ui;
}
/ To change font color and content background /
.e-richtexteditor .e-rte-content,
.e-richtexteditor .e-source-content {
background: seashell;
color: blue;
}
`
```

[Customizing the Rich Text Editor's toolbar](#)

Use the following CSS to customize the default color in the Rich Text Editor's toolbar icon.

```
`css
/ To change font color for toolbar icon /
.e-richtexteditor .e-rte-toolbar .e-toolbar-item .e-icons,
.e-richtexteditor .e-rte-toolbar .e-toolbar-item .e-icons:active {
color: red;
}
/ To change font color for toolbar button /
.e-toolbar .e-tbar-btn,
.e-toolbar .e-tbar-btn:active,
.e-toolbar .e-tbar-btn:hover {
```

```
color: red;
}

/ To change font color for toolbar button in active state/
.e-richtexteditor .e-rte-toolbar .e-toolbar-item .e-dropdown-btn.e-active .e-icons, .e-richtexteditor .e-
rte-toolbar .e-toolbar-item .e-dropdown-btn.e-active .e-rte-dropdown-btn-text {
color: red;
}

/ To change font color for expanded toolbar items /
.e-richtexteditor .e-rte-toolbar .e-toolbar-extended .e-toolbar-item .e-tbar-btn .e-icons,
.e-toolbar.e-extended-toolbar .e-toolbar-extended .e-toolbar-item .e-tbar-btn {
color: red;
}
`
```

Customizing the Rich Text Editor's character count

Use the following CSS to customize the default color in the Rich Text Editor's character count.

```
`css

/ To change font color, font family, font size and opacity /
.e-richtexteditor .e-rte-character-count {
color: red;
font-family: segoe ui;
font-size: 18px;
opacity: 00.54;
padding-bottom: 2px;
padding-right: 14px;
}
`
```

Keyboard support in React Rich text editor component

The editor has full keyboard accessibility that includes shortcuts to open and other actions with toolbar items, drop-down lists, and dialogs.

HTML formation shortcut key

You can use the following key shortcuts when the Rich Text Editor renders with HTML edit mode.

| Actions | Keyboard shortcuts |

|-----|-----|

| **Toolbar focus | Alt + f10 |**

| Insert link | Ctrl + k |

| Insert image | Ctrl + Shift + i |

| Insert audio | Ctrl + Shift + a |

| Insert video | Ctrl + Alt + v |

| Insert table | Ctrl + Shift + e |

| Undo | Ctrl + z |

| Redo | Ctrl + y |

| Copy | Ctrl + c |

| Cut | Ctrl + x |

| Paste | Ctrl + v |

| Bold | Ctrl + b |

| Italic | Ctrl + i |

| Underline | Ctrl + u |

| Strikethrough | Ctrl + Shift + s |

| Uppercase | Ctrl + Shift + u |

| Lowercase | Ctrl + Shift + l |

| Superscript | Ctrl + Shift + = |

| Subscript | Ctrl + = |

| Indents | Ctrl +] |

| Outdents | Ctrl + [|

| HTML source | Ctrl + Shift + h |

| Fullscreen | Ctrl + Shift + f |

| Exit Fullscreen | Esc |

| Justify center | Ctrl + e |

| Justify full | Ctrl + j |

| Justify left | Ctrl + l |

| Justify right | Ctrl + r |

| Clear format | Ctrl + Shift + r |

| Ordered list | Ctrl + Shift + o |

| Unordered list | Ctrl + Alt + o |

| Format Painter Copy | Alt + Shift + c |

| Format Painter Paste | Alt + Shift + v |

| Format Painter Escape | Esc |

[Class-component]

APP.JSX

```
{% raw %}
/**
 * Rich Text Editor - HTMLEditor KeyConfig sample
 */
import { RichTextEditorComponent, Inject, Toolbar, HtmlEditor, Image,
QuickToolbar, Link, FormatPainter} from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'FormatPainter', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  }
  docKeyUp(e) {
    if (e.altKey && e.keyCode === 84) { /* t */
      // press alt+t to focus the component.
      this.rteObj.focusIn();
    }
  }
  componentDidMount() {
    document.addEventListener('keyup', this.docKeyUp.bind(this));
  }
  render() {
    return (<RichTextEditorComponent ref={ (richtexteditor) => {
this.rteObj = richtexteditor; }} height={450}
toolbarSettings={this.toolbarSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>

```

```

        <p>Provides a fully customizable toolbar.</p>
    </li>
    <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
    </li>
    <li>
        <p>Supports third-party library integration.</p>
    </li>
    <li>
        <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
    </li>
    <li>
        <p>Contains undo/redo manager.</p>
    </li>
    <li>
        <p>Creates bulleted and numbered lists.</p>
    </li>
</ul>
    <Inject services={[Toolbar, HtmlEditor, Image, QuickToolbar, Link,
FormatPainter]} />
    </RichTextEditorComponent>);
    }
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - HTMLEditor KeyConfig sample
 */
import { RichTextEditorComponent, Inject, Toolbar, HtmlEditor, Image,
QuickToolbar, Link, FormatPainter} from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{},{}> {
    private rteObj: RichTextEditorComponent;
    private toolbarSettings: object = {
        items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'FormatPainter', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
    }
    public docKeyUp(e: any) {
        if (e.altKey && e.keyCode === 84) { /* t */
            // press alt+t to focus the component.
            this.rteObj.focusIn();
        }
    }
}

```

```

    }
  }
  public componentDidMount() {
    document.addEventListener('keyup', this.docKeyUp.bind(this));
  }
  public render() {
    return (
      <RichTextEditorComponent ref={(richtexteditor) => { this.rteObj =
richtexteditor! }} height={450} toolbarSettings={this.toolbarSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
          Users can format their content using standard toolbar commands.</p>
          <p><b>Key features:</b></p>
          <ul>
            <li>
              <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
              <p>Capable of handling markdown editing.</p>
            </li>
            <li>
              <p>Contains a modular library to load the necessary functionality
on demand.</p>
            </li>
            <li>
              <p>Provides a fully customizable toolbar.</p>
            </li>
            <li>
              <p>Provides HTML view to edit the source directly for
developers.</p>
            </li>
            <li>
              <p>Supports third-party library integration.</p>
            </li>
            <li>
              <p>Allows preview of modified content before saving it.</p>
            </li>
            <li>
              <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
            </li>
            <li>
              <p>Contains undo/redo manager.</p>
            </li>
            <li>
              <p>Creates bulleted and numbered lists.</p>
            </li>
          </ul>
          <Inject services={[Toolbar, HtmlEditor, Image, QuickToolbar, Link,
FormatPainter]} />
        </RichTextEditorComponent>
      );
    );
  }
}
export default App;
{% endraw %}

```


[Functional-component]**APP.JSX**

```
{% raw %}
/**
 * Rich Text Editor - HTMLEditor KeyConfig sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let rteObj;
  let toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  function docKeyUp(e) {
    if (e.altKey && e.keyCode === 84) { /* t */
      // press alt+t to focus the component.
      rteObj.focusIn();
    }
  }
  function componentDidMount() {
    document.addEventListener('keyup', docKeyUp.bind(this));
  }
  return (<RichTextEditorComponent ref={rteObj} height={450} toolbarSettings={toolbarSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
    get") editor that provides the best user experience to create and update the
    content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
        on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
    </ul>
  </RichTextEditorComponent>);
}
```

```

        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
  </RichTextEditorComponent>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - HTMLEditor KeyConfig sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let rteObj: RichTextEditorComponent;
  let toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  }
  function docKeyUp(e: any) {
    if (e.altKey && e.keyCode === 84) { /* t */
      // press alt+t to focus the component.
      rteObj.focusIn();
    }
  }
  function componentDidMount() {
    document.addEventListener('keyup', docKeyUp.bind(this));
  }
  return (

```

```

<RichTextEditorComponent ref={(richtexteditor) => {rteObj =
richtexteditor! }} height={450} toolbarSettings={toolbarSettings}>
  <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
  Users can format their content using standard toolbar commands.</p>
  <p><b>Key features:</b></p>
  <ul>
    <li>
      <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
    </li>
    <li>
      <p>Capable of handling markdown editing.</p>
    </li>
    <li>
      <p>Contains a modular library to load the necessary functionality
on demand.</p>
    </li>
    <li>
      <p>Provides a fully customizable toolbar.</p>
    </li>
    <li>
      <p>Provides HTML view to edit the source directly for
developers.</p>
    </li>
    <li>
      <p>Supports third-party library integration.</p>
    </li>
    <li>
      <p>Allows preview of modified content before saving it.</p>
    </li>
    <li>
      <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>
    </li>
    <li>
      <p>Contains undo/redo manager.</p>
    </li>
    <li>
      <p>Creates bulleted and numbered lists.</p>
    </li>
  </ul>
  <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
export default App;
{% endraw %}

```

Markdown formation shortcut key

You can use the following key shortcuts when the Rich Text Editor renders with Markdown edit mode

| Actions | Keyboard shortcuts |

|-----|-----|

| **Toolbar focus** | **Alt + f10** |

| Insert link| Ctrl + k |

| Insert image| Ctrl + Shift + i |

| Insert table| Ctrl + Shift + e |

| Undo| Ctrl + z |

| Redo| Ctrl + y |

| Copy| Ctrl + c |

| Cut| Ctrl + x |

| Paste| Ctrl + v |

| Bold| Ctrl + b |

| Italic| Ctrl + i |

| Strikethrough| Ctrl + Shift + s |

| Uppercase| Ctrl + Shift + u |

| Lowercase| Ctrl + Shift + l |

| Superscript| Ctrl + Shift + = |

| Subscript| Ctrl + = |

| Fullscreen| Ctrl + Shift + f |

| Ordered list| Ctrl + Shift + o |

| Unordered list| Ctrl + Alt + o |

[Class-component]

APP.JSX

```
{% raw %}
/**
 * Rich Text Editor - MarkdownEditor KeyConfig sample
 */
import { Image, Inject, Link, MarkdownEditor, RichTextEditorComponent,
Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  toolbarSettings = {
    items: ['Bold', 'Italic', 'StrikeThrough', '|',
      'Formats', 'OrderedList', 'UnorderedList', '|',
      'CreateLink', 'Image', '|', 'Undo', 'Redo']
  };
  valueTemplate() {
    return (<div>
      The sample is added to showcase **markdown editing**.
      Type or edit the content and apply formatting to view markdown formatted
      content.
    </div>);
  }
}
```

```

    We can add our own custom formation syntax for the Markdown formation,
    [sample link] (https://ej2.syncfusion.com/home/).
    The third-party library <b>Marked</b> is used in this sample to convert
    markdown into HTML content.
    </div>);
  }
;
componentDidMount() {
  document.addEventListener('keyup', this.docKeyUp.bind(this));
}
docKeyUp(e) {
  if (e.altKey && e.keyCode === 84) { /* t */
    // press alt+t to focus the component.
    this.rteObj.focusIn();
  }
}
render() {
  return (<RichTextEditorComponent ref={(richtexteditor) => {
    this.rteObj = richtexteditor; }} height={450}
    toolbarSettings={this.toolbarSettings} valueTemplate={this.valueTemplate}
    editorMode={'Markdown'}>
    <Inject services={[Toolbar, Image, Link, MarkdownEditor]}/>
    </RichTextEditorComponent>);
}
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - MarkdownEditor KeyConfig sample
 */
import { Image, Inject, Link, MarkdownEditor, RichTextEditorComponent,
Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private rteObj: RichTextEditorComponent;
  private toolbarSettings: object = {
    items: ['Bold', 'Italic', 'StrikeThrough', '|',
      'Formats', 'OrderedList', 'UnorderedList', '|',
      'CreateLink', 'Image', '|', 'Undo', 'Redo']
  }
  public valueTemplate(): JSX.Element {
    return (<div>
      The sample is added to showcase **markdown editing**.
      Type or edit the content and apply formatting to view markdown formatted
      content.
      We can add our own custom formation syntax for the Markdown formation,
      [sample link] (https://ej2.syncfusion.com/home/).
      The third-party library <b>Marked</b> is used in this sample to convert
      markdown into HTML content.
      </div>);
  };
  public componentDidMount() {

```

```

document.addEventListener('keyup', this.docKeyUp.bind(this));
}
public docKeyUp(e: any) {
  if (e.altKey && e.keyCode === 84) { /* t */
    // press alt+t to focus the component.
    this.rteObj.focusIn();
  }
}
public render() {
  return (
    <RichTextEditorComponent ref={(richtexteditor) => { this.rteObj =
richtexteditor! }} height={450} toolbarSettings={this.toolbarSettings}
valueTemplate={this.valueTemplate} editorMode='Markdown'>
      <Inject services={[Toolbar, Image, Link, MarkdownEditor]} />
    </RichTextEditorComponent>
  );
}
}
export default App;
{% enddraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - MarkdownEditor KeyConfig sample
 */
import { Image, Inject, Link, MarkdownEditor, RichTextEditorComponent,
Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  let rteObj;
  let toolbarSettings = {
    items: ['Bold', 'Italic', 'StrikeThrough', '|',
      'Formats', 'OrderedList', 'UnorderedList', '|',
      'CreateLink', 'Image', '|', 'Undo', 'Redo']
  };
  function valueTemplate() {
    return (<div>
      The sample is added to showcase **markdown editing**.
      Type or edit the content and apply formatting to view markdown
formatted content.
      We can add our own custom formation syntax for the Markdown formation,
[sample link](https://ej2.syncfusion.com/home/).
      The third-party library <b>Marked</b> is used in this sample to convert
markdown into HTML content.
    </div>);
  }
  ;
  function componentDidMount() {
    document.addEventListener('keyup', docKeyUp.bind(this));
  }
  function docKeyUp(e) {
    if (e.altKey && e.keyCode === 84) { /* t */

```

```

        // press alt+t to focus the component.
        rteObj.focusIn();
    }
}
return (<RichTextEditorComponent ref={(richtexteditor) => { rteObj =
richtexteditor; }} height={450} toolbarSettings={toolbarSettings}
valueTemplate={valueTemplate} editorMode={'Markdown'}>
    <Inject services={[Toolbar, Image, Link, MarkdownEditor]}/>
</RichTextEditorComponent>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - MarkdownEditor KeyConfig sample
 */
import { Image, Inject, Link, MarkdownEditor, RichTextEditorComponent,
Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
    let rteObj: RichTextEditorComponent;
    let toolbarSettings: object = {
        items: ['Bold', 'Italic', 'StrikeThrough', '|',
            'Formats', 'OrderedList', 'UnorderedList', '|',
            'CreateLink', 'Image', '|', 'Undo', 'Redo']
    }
    function valueTemplate(): JSX.Element {
        return(<div>
            The sample is added to showcase **markdown editing**.
            Type or edit the content and apply formatting to view markdown
            formatted content.
            We can add our own custom formation syntax for the Markdown formation,
            [sample link] (https://ej2.syncfusion.com/home/).
            The third-party library <b>Marked</b> is used in this sample to convert
            markdown into HTML content.
        </div>);
    };
    function componentDidMount() {
        document.addEventListener('keyup', docKeyUp.bind(this));
    }
    function docKeyUp(e: any) {
        if (e.altKey && e.keyCode === 84) { /* t */
            // press alt+t to focus the component.
            rteObj.focusIn();
        }
    }
    return (
        <RichTextEditorComponent ref={(richtexteditor) => { rteObj =
            richtexteditor! }} height={450} toolbarSettings={toolbarSettings}
            valueTemplate={valueTemplate} editorMode={'Markdown'}>
            <Inject services={[Toolbar, Image, Link, MarkdownEditor]}/>
        </RichTextEditorComponent>
    );
}

```

```

}
export default App;
{% endraw %}

```

Custom key config

Customize the key config for the keyboard interaction of Rich Text Editor, using the [keyConfig](#) property.

In the following sample, customize the cut, copy, paste toolbar action with ctrl+1, ctrl+2, ctrl+3, respectively.

[Class-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - Custom KeyConfig Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  keyConfig = {
    'copy': 'ctrl+1',
    'cut': 'ctrl+2',
    'paste': 'ctrl+3'
  };
  docKeyUp(e) {
    if (e.altKey && e.keyCode === 84) { /* t */
      // press alt+t to focus the component.
      this.rteObj.focusIn();
    }
  }
  componentDidMount() {
    document.addEventListener('keyup', this.docKeyUp.bind(this));
  }
  render() {
    return (
      <RichTextEditorComponent ref={ (richtexteditor) => {
        this.rteObj = richtexteditor; }} height={450}
        toolbarSettings={this.toolbarSettings} keyConfig={this.keyConfig}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
        you get") editor that provides the best user experience to create and update
        the content. Users can format their content using standard toolbar
        commands.</p>
        <p><b>Key features:</b></p>

```



```

        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
          </li>
          <li>
            <p>Contains undo/redo manager.</p>
          </li>
          <li>
            <p>Creates bulleted and numbered lists.</p>
          </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
      </RichTextEditorComponent>;
    }
  }
  export default App;
  {% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Custom KeyConfig Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private rteObj: RichTextEditorComponent;
  private toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',

```

```

        'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
        'LowerCase', 'UpperCase', '|',
        'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
        'Outdent', 'Indent', '|',
        'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
        'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
    }
    private keyConfig: object = {
        'copy': 'ctrl+1',
        'cut': 'ctrl+2',
        'paste': 'ctrl+3'
    }
    public docKeyUp(e: any) {
        if (e.altKey && e.keyCode === 84) { /* t */
            // press alt+t to focus the component.
            this.rteObj.focusIn();
        }
    }
    public componentDidMount() {
        document.addEventListener('keyup', this.docKeyUp.bind(this));
    }
    public render() {
        return (
            <RichTextEditorComponent ref={(richtexteditor) => { this.rteObj =
richtexteditor! }} height={450} toolbarSettings={this.toolbarSettings}
keyConfig={this.keyConfig}>
                <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
                <p><b>Key features:</b></p>
                <ul>
                    <li>
                        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
                    </li>
                    <li>
                        <p>Capable of handling markdown editing.</p>
                    </li>
                    <li>
                        <p>Contains a modular library to load the necessary functionality
on demand.</p>
                    </li>
                    <li>
                        <p>Provides a fully customizable toolbar.</p>
                    </li>
                    <li>
                        <p>Provides HTML view to edit the source directly for
developers.</p>
                    </li>
                    <li>
                        <p>Supports third-party library integration.</p>
                    </li>
                    <li>
                        <p>Allows preview of modified content before saving it.</p>
                    </li>
                </ul>
            </RichTextEditorComponent>
        )
    }

```

```

        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
  </RichTextEditorComponent>
);
}
}
export default App;
{% endraw %}

```

[Functional-component]

APP.JSX

```

{% raw %}
/**
 * Rich Text Editor - Custom KeyConfig Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let rteObj;
  let toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  let keyConfig = {
    'copy': 'ctrl+1',
    'cut': 'ctrl+2',
    'paste': 'ctrl+3'
  };
  function docKeyUp(e) {
    if (e.altKey && e.keyCode === 84) { /* t */
      // press alt+t to focus the component.
      rteObj.focusIn();
    }
  }
  function componentDidMount() {
    document.addEventListener('keyup', docKeyUp.bind(this));
  }
}

```

```

    return (<RichTextEditorComponent ref={(richtexteditor) => { rteObj =
richtexteditor; }} height={450} toolbarSettings={toolbarSettings}
keyConfig={keyConfig}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>
    </RichTextEditorComponent>);
  }
  export default App;
  {% endraw %}

```

APP.TSX

```

{% raw %}
/**
 * Rich Text Editor - Custom KeyConfig Sample
 */

```

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  let rteObj: RichTextEditorComponent;
  let toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  }
  let keyConfig: object = {
    'copy': 'ctrl+1',
    'cut': 'ctrl+2',
    'paste': 'ctrl+3'
  }
  function docKeyUp(e: any) {
    if (e.altKey && e.keyCode === 84) { /* t */
      // press alt+t to focus the component.
      rteObj.focusIn();
    }
  }
  function componentDidMount() {
    document.addEventListener('keyup', docKeyUp.bind(this));
  }
  return (
    <RichTextEditorComponent ref={(richtexteditor) => { rteObj =
richtexteditor! }} height={450} toolbarSettings={toolbarSettings}
keyConfig={keyConfig}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
      </ul>
    </RichTextEditorComponent>
  );
}

```

```

        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
  />
</RichTextEditorComponent>
);
}
export default App;
{% enddraw %}

```

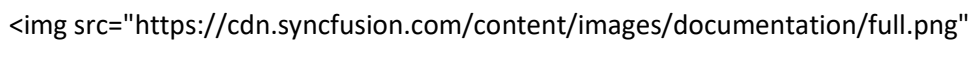
Accessibility in React Rich text editor component

The Rich Text Editor component has been designed, keeping in mind the WAI-ARIA specifications, and applies the WAI-ARIA roles, states, and properties. This component is characterized by complete ARIA accessibility support that makes it easy for people who use assistive technologies (AT) or those who completely rely on keyboard navigation.

The accessibility compliance for the Rich Text Editor component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  alt="Yes"> |

| [Section 508 Support](#) |  alt="Yes"> |

| [Screen Reader Support](#) |  alt="Yes"> |

| [Right-To-Left Support](#) |  alt="Yes"> |

| [Color Contrast](#) |  alt="Intermediate"> |

| [Mobile Device Support](#) |  alt="Yes"> |

| [Keyboard Navigation Support](#) |  alt="Yes"> |

```
| Accessibility Checker Validation |  |
| Axe-core Accessibility Validation |  |

<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>
```

ARIA attributes

The toolbar of Rich Text Editor, assigned the role of Toolbar and has the following list of ARIA attributes.

| Roles and Attributes | Functionalities |

| --- | --- |

| role="toolbar" | This attribute added to the toolbar element describes the actual role of the element. |

| aria-orientation | Indicates the toolbar orientation. Default value is horizontal. |

| aria-haspopup | Indicates the popup mode of the toolbar. The default value is false. When popup mode is enabled, attribute value has to be changed to true. |

| aria-disabled | Indicates the disabled state of the toolbar. |

| aria-owns | Identifies an element to define a visual, functional, or contextual parent/child relationship between DOM elements when the DOM hierarchy cannot represent the relationship. In the Rich Text Editor, the attribute contains the ID of the Rich Text Editor to indicate the popup as a child element. |

For further details of toolbar ARIA attributes, refer the accessibility of [Toolbar](#) documentation.

The Rich Text Editor element is assigned the role of application.

| Roles and Attributes | Functionalities |

| --- | --- |

| role="application" | This attribute added to the Rich Text Editor element describes the actual role of the element. |

| aria-disabled | Indicates the disabled state of the Rich Text Editor. |

[Class-component]**APP.JSX**

```

/**
 * Rich Text Editor - Accessibility Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  iframeSettings = {
    enable: true
  };
  render() {
    return (<RichTextEditorComponent height={450}
iframeSettings={this.iframeSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
      <li>
        <p>Creates bulleted and numbered lists.</p>
      </li>
    </ul>
  </RichTextEditorComponent>
  </App>
  </pre>

```



```

        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
      </RichTextEditorComponent>);
    }
  }
  export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Accessibility Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private iframeSettings: object = {
    enable: true
  }
  public render() {
    return (
      <RichTextEditorComponent height={450}
iframeSettings={this.iframeSettings}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
          <li>
            <p>Provides HTML view to edit the source directly for
developers.</p>
          </li>
          <li>
            <p>Supports third-party library integration.</p>
          </li>
          <li>
            <p>Allows preview of modified content before saving it.</p>
          </li>
          <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>

```

```

        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
    />
  </RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

/**
 * Rich Text Editor - Accessibility Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let iframeSettings = {
    enable: true
  };
  return (<RichTextEditorComponent height={450}
iframeSettings={iframeSettings}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
    </ul>

```

```

        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>/>
    </RichTextEditorComponent>;
  }
  export default App;

```

APP.TSX

```

/**
 * Rich Text Editor - Accessibility Sample
 */
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let iframeSettings: object = {
    enable: true
  }

  return (
    <RichTextEditorComponent height={450} iframeSettings={iframeSettings}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content. Users can format their content using standard toolbar
commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>

```

```

        </li>
        <li>
          <p>Provides HTML view to edit the source directly for
developers.</p>
        </li>
        <li>
          <p>Supports third-party library integration.</p>
        </li>
        <li>
          <p>Allows preview of modified content before saving it.</p>
        </li>
        <li>
          <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
          <p>Contains undo/redo manager.</p>
        </li>
        <li>
          <p>Creates bulleted and numbered lists.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
    </RichTextEditorComponent>
  );
}
export default App;

```

Keyboard interaction

The Rich Text Editor component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Rich Text Editor component.

HTML formation shortcut key

You can use the following key shortcuts when the Rich Text Editor renders with HTML edit mode.

| Actions | Keyboard shortcuts |

|-----|-----|

| Toolbar focus | Alt + f10 |

| Insert link | Ctrl + k |

| Insert image | Ctrl + Shift + i |

| Insert audio | Ctrl + Shift + a |

| Insert video | Ctrl + Alt + v |

| Insert table | Ctrl + Shift + e |

| Undo | Ctrl + z |

| Redo | Ctrl + y |

Copy	Ctrl + c
Cut	Ctrl + x
Paste	Ctrl + v
Bold	Ctrl + b
Italic	Ctrl + i
Underline	Ctrl + u
Strikethrough	Ctrl + Shift + s
Uppercase	Ctrl + Shift + u
Lowercase	Ctrl + Shift + l
Superscript	Ctrl + Shift + =
Subscript	Ctrl + =
Indents	Ctrl +]
Outdents	Ctrl + [
HTML source	Ctrl + Shift + h
Fullscreen	Ctrl + Shift + f
Exit Fullscreen	Esc
Justify center	Ctrl + e
Justify full	Ctrl + j
Justify left	Ctrl + l
Justify right	Ctrl + r
Clear format	Ctrl + Shift + r
Ordered list	Ctrl + Shift + o
Unordered list	Ctrl + Alt + o
Format Painter Copy	Alt + Shift + c
Format Painter Paste	Alt + Shift + v
Format Painter Escape	Esc

Markdown formation shortcut key

You can use the following key shortcuts when the Rich Text Editor renders with Markdown edit mode

Actions	Keyboard shortcuts
Toolbar focus	Alt + f10
Insert link	Ctrl + k

Insert image	Ctrl + Shift + i
Insert table	Ctrl + Shift + e
Undo	Ctrl + z
Redo	Ctrl + y
Copy	Ctrl + c
Cut	Ctrl + x
Paste	Ctrl + v
Bold	Ctrl + b
Italic	Ctrl + i
Strikethrough	Ctrl + Shift + s
Uppercase	Ctrl + Shift + u
Lowercase	Ctrl + Shift + l
Superscript	Ctrl + Shift + =
Subscript	Ctrl + =
Fullscreen	Ctrl + Shift + f
Ordered list	Ctrl + Shift + o
Unordered list	Ctrl + Alt + o

Ensuring accessibility

The Rich Text Editor component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Rich Text Editor component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Rich Text Editor component with accessibility tools.

See also

- [Accessibility in Syncfusion React components](#)

How To

Add google fonts in React Rich text editor component

To use web fonts in RTE, it is not needed for the web fonts to be present in local machine. To add the web fonts to RTE, we need to refer the web font links and add the font names in the [fontFamily](#) property.

[Class-component]

APP.JSX

```

import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  fontFamily = {
    items: [
      { text: "Segoe UI", value: "Segoe UI", class: "e-segoe-ui",
command: "Font", subCommand: "FontName" },
      { text: "Roboto", value: "Roboto", command: "Font", subCommand:
"FontName" },
      { text: "Great vibes", value: "Great Vibes,cursive", command:
"Font", subCommand: "FontName" },
      { text: "Noto Sans", value: "Noto Sans", command: "Font",
subCommand: "FontName" },
      { text: "Impact", value: "Impact,Charcoal,sans-serif", class: "e-
impact", command: "Font", subCommand: "FontName" },
      { text: "Tahoma", value: "Tahoma,Geneva,sans-serif", class: "e-
tahoma", command: "Font", subCommand: "FontName" },
    ]
  };
  toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  render() {
    return (<RichTextEditorComponent
toolbarSettings={this.toolbarSettings} fontFamily={this.fontFamily}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]}/>
    </RichTextEditorComponent>);
  }
}

```

```
}
export default App;
```

APP.TSX

```
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private fontFamily: object = {
    items: [
      {text: "Segoe UI", value: "Segoe UI", class: "e-segoe-ui",
command: "Font", subCommand: "FontName"},
      {text: "Roboto", value: "Roboto", command: "Font", subCommand:
"FontName"},
      {text: "Great vibes", value: "Great Vibes, cursive", command:
"Font", subCommand: "FontName"},
      {text: "Noto Sans", value: "Noto Sans", command: "Font",
subCommand: "FontName"},
      {text: "Impact", value: "Impact, Charcoal, sans-serif", class: "e-
impact", command: "Font", subCommand: "FontName"},
      {text: "Tahoma", value: "Tahoma, Geneva, sans-serif", class: "e-
tahoma", command: "Font", subCommand: "FontName"},
    ]
  };
  private toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  public render() {
    return (
      <RichTextEditorComponent toolbarSettings={this.toolbarSettings}
fontFamily={this.fontFamily}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
```



```

        <p>Provides a fully customizable toolbar.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]} />
  </RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  const fontFamily = {
    items: [
      { text: "Segoe UI", value: "Segoe UI", class: "e-segoe-ui",
command: "Font", subCommand: "FontName" },
      { text: "Roboto", value: "Roboto", command: "Font", subCommand:
"FontName" },
      { text: "Great vibes", value: "Great Vibes,cursive", command:
"Font", subCommand: "FontName" },
      { text: "Noto Sans", value: "Noto Sans", command: "Font",
subCommand: "FontName" },
      { text: "Impact", value: "Impact,Charcoal,sans-serif", class: "e-
impact", command: "Font", subCommand: "FontName" },
      { text: "Tahoma", value: "Tahoma, Geneva, sans-serif", class: "e-
tahoma", command: "Font", subCommand: "FontName" },
    ]
  };
  const toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  return (
    <RichTextEditorComponent toolbarSettings={toolbarSettings}
fontFamily={fontFamily}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>

```

```

        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]}/>
    </RichTextEditorComponent>);
  }
  export default App;

```

APP.TSX

```

import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  const fontFamily: object = {
    items: [
      {text: "Segoe UI", value: "Segoe UI", class: "e-segoe-ui", command:
"Font", subCommand: "FontName"},
      {text: "Roboto", value: "Roboto", command: "Font", subCommand:
"FontName"},
      {text: "Great vibes", value: "Great Vibes,cursive", command: "Font",
subCommand: "FontName"},
      {text: "Noto Sans", value: "Noto Sans", command: "Font", subCommand:
"FontName"},
      {text: "Impact", value: "Impact,Charcoal,sans-serif", class: "e-
impact", command: "Font", subCommand: "FontName"},
      {text: "Tahoma", value: "Tahoma,Geneva,sans-serif", class: "e-tahoma",
command: "Font", subCommand: "FontName"},
    ]
  };
  const toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  return (
    <RichTextEditorComponent toolbarSettings={toolbarSettings}
fontFamily={fontFamily}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
    </RichTextEditorComponent>
  );
}

```

```

    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]} />
  </RichTextEditorComponent>
);
}
export default App;

```

The below font style links are referred in the page.

`ts

<link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Roboto">

<link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Great+Vibes">

,

In the above sample, you can see that we have added two Google web fonts (Roboto and Great vibes) to RTE.

Change default font family in React Rich text editor component

By using [default](#) property, you can change the default font-family of the RTE. To change the font-family of the RTE content while loading, we need to give the font-family in the style section with the help of [cssClass](#) property.

[Class-component]

APP.JSX

```

import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  fontFamily = {
    default: "Noto Sans",
    items: [
      { text: "Segoe UI", value: "Segoe UI", class: "e-segoe-ui",
command: "Font", subCommand: "FontName" },
      { text: "Roboto", value: "Roboto", command: "Font", subCommand:
"FontName" },

```

```

        { text: "Great vibes", value: "Great Vibes,cursive", command:
"Font", subCommand: "FontName" },
        { text: "Noto Sans", value: "Noto Sans", command: "Font",
subCommand: "FontName" },
        { text: "Impact", value: "Impact,Charcoal,sans-serif", class: "e-
impact", command: "Font", subCommand: "FontName" },
        { text: "Tahoma", value: "Tahoma,Geneva,sans-serif", class: "e-
tahoma", command: "Font", subCommand: "FontName" },
    ]
    };
    toolbarSettings = {
      items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'Fullscreen', '|', 'Undo', 'Redo']
    };
    cssClass = "customClass";
    render() {
      return (<RichTextEditorComponent
toolbarSettings={this.toolbarSettings} fontFamily={this.fontFamily}
cssClass={this.cssClass}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
        </ul>
        <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]}/>
      </RichTextEditorComponent>);
    }
  }
  export default App;

```

APP.TSX

```

import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';

```

```

import * as React from 'react';
class App extends React.Component<{}> {
  private fontFamily: object = {
    default: "Noto Sans", // to define default font-family
    items: [
      {text: "Segoe UI", value: "Segoe UI", class: "e-segoe-ui",
command: "Font", subCommand: "FontName"},
      {text: "Roboto", value: "Roboto", command: "Font", subCommand:
"FontName"},
      {text: "Great vibes", value: "Great Vibes,cursive", command:
"Font", subCommand: "FontName"},
      {text: "Noto Sans", value: "Noto Sans", command: "Font",
subCommand: "FontName"},
      {text: "Impact", value: "Impact,Charcoal,sans-serif", class: "e-
impact", command: "Font", subCommand: "FontName"},
      {text: "Tahoma", value: "Tahoma,Geneva,sans-serif", class: "e-
tahoma", command: "Font", subCommand: "FontName"},
    ]
  };
  private toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  private cssClass: string = "customClass";
  public render() {
    return (
      <RichTextEditorComponent toolbarSettings={this.toolbarSettings}
fontFamily={this.fontFamily} cssClass={this.cssClass}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
          <li>
            <p>Contains a modular library to load the necessary functionality
on demand.</p>
          </li>
          <li>
            <p>Provides a fully customizable toolbar.</p>
          </li>
        </ul>
        <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]} />
      </RichTextEditorComponent>
    );
  }
}

```

```

    }
  }
  export default App;

```

[Functional-component]

APP.JSX

```

import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  const fontFamily = {
    default: "Noto Sans",
    items: [
      { text: "Segoe UI", value: "Segoe UI", class: "e-segoe-ui",
command: "Font", subCommand: "FontName" },
      { text: "Roboto", value: "Roboto", command: "Font", subCommand:
"FontName" },
      { text: "Great vibes", value: "Great Vibes,cursive", command:
"Font", subCommand: "FontName" },
      { text: "Noto Sans", value: "Noto Sans", command: "Font",
subCommand: "FontName" },
      { text: "Impact", value: "Impact,Charcoal,sans-serif", class: "e-
impact", command: "Font", subCommand: "FontName" },
      { text: "Tahoma", value: "Tahoma,Geneva,sans-serif", class: "e-
tahoma", command: "Font", subCommand: "FontName" },
    ]
  };
  const toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  let cssClass = "customClass";
  return (<RichTextEditorComponent toolbarSettings={toolbarSettings}
fontFamily={fontFamily} cssClass={cssClass}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>

```

```

        <p>Contains a modular library to load the necessary functionality
on demand.</p>
    </li>
    <li>
        <p>Provides a fully customizable toolbar.</p>
    </li>
</ul>
    <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]}/>
    </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    const fontFamily: object = {
        default: "Noto Sans", // to define default font-family
        items: [
            {text: "Segoe UI", value: "Segoe UI", class: "e-segoe-ui", command:
"Font", subCommand: "FontName"},
            {text: "Roboto", value: "Roboto", command: "Font", subCommand:
"FontName"},
            {text: "Great vibes", value: "Great Vibes,cursive", command: "Font",
subCommand: "FontName"},
            {text: "Noto Sans", value: "Noto Sans", command: "Font", subCommand:
"FontName"},
            {text: "Impact", value: "Impact,Charcoal,sans-serif", class: "e-
impact", command: "Font", subCommand: "FontName"},
            {text: "Tahoma", value: "Tahoma,Geneva,sans-serif", class: "e-tahoma",
command: "Font", subCommand: "FontName"},
        ]
    };
    const toolbarSettings: object = {
        items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
    };

    let cssClass: string = "customClass";
    return (
        <RichTextEditorComponent toolbarSettings={toolbarSettings}
fontFamily={fontFamily} cssClass={cssClass}>
            <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
                Users can format their content using standard toolbar commands.</p>
            <p><b>Key features:</b></p>

```

```

    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]} />
  </RichTextEditorComponent>
);
}
export default App;

```

Check image size in React Rich text editor component

By using the Rich text editor's `imageUploading` event, you can get the image size before uploading and restrict the image to upload, when the given image size is greater than the allowed size.

In the following, we have validated the image size before uploading and determined whether the image has been uploaded or not.

[Class-component]

APP.JSX

```

import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
export class App extends React.Component {
  toolbarSettings = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  insertImageSettings = {
    saveUrl:
      "https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save",
    path: "../Images/"
  };
  onImageUploading = (args) => {
    console.log("file is uploading");
    let imgSize = 500000;
    let sizeInBytes = args.fileData.size;

```



```

        if (imgSize < sizeInBytes) {
            args.cancel = true;
        }
    };
    render() {
        return (<RichTextEditorComponent
toolbarSettings={this.toolbarSettings}
insertImageSettings={this.insertImageSettings}
imageUploading={this.onImageUploading.bind(this)}>
        <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]} />
        </RichTextEditorComponent>);
    }
}
export default App;

```

APP.TSX

```

import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
export class App extends React.Component<{},> {
    private toolbarSettings: object = {
        items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
    };
    private insertImageSettings: object = {
        saveUrl: "https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save",
        path: "../Images/"
    };
    public onImageUploading = (args: any) => {
        console.log("file is uploading");
        let imgSize: number = 500000;
        let sizeInBytes: number = args.fileData.size;
        if (imgSize < sizeInBytes) {
            args.cancel = true;
        }
    }
    public render() {
        return (
            <RichTextEditorComponent toolbarSettings={this.toolbarSettings}
insertImageSettings={this.insertImageSettings}
imageUploading={this.onImageUploading.bind(this)} >
                <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]} />
                </RichTextEditorComponent>
        );
    }
}
export default App;

```

[Functional-component]**APP.JSX**

```

import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
    const toolbarSettings = {
        items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
            'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
            'LowerCase', 'UpperCase', '|',
            'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
            'Outdent', 'Indent', '|',
            'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
            'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
    };
    const insertImageSettings = {
        saveUrl:
"https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save",
        path: "../Images/"
    };
    function onImageUploading(args) {
        console.log("file is uploading");
        let imgSize = 500000;
        let sizeInBytes = args.fileData.size;
        if (imgSize < sizeInBytes) {
            args.cancel = true;
        }
    }
    return (<RichTextEditorComponent toolbarSettings={toolbarSettings}
insertImageSettings={insertImageSettings}
imageUploading={onImageUploading.bind(this)}>
        <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]}/>
        </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App(){
    const toolbarSettings: object = {
        items: ['Bold', 'Italic', 'Underline', 'StrikeThrough','|',
            'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
            'LowerCase', 'UpperCase', '|',
            'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
            'Outdent', 'Indent', '|',
            'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
            'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
    }

```

```

    };
    const insertImageSettings: object = {
      saveUrl:
        "https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save",
      path: "../Images/"
    };
    function onImageUploading(args: any){
      console.log("file is uploading");
      let imgSize: number = 500000;
      let sizeInBytes: number = args.fileData.size;
      if ( imgSize < sizeInBytes ) {
        args.cancel = true;
      }
    }
    return (
      <RichTextEditorComponent toolbarSettings={toolbarSettings}
        insertImageSettings={insertImageSettings}
        imageUploading={onImageUploading.bind(this)} >
        <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
        QuickToolbar]} />
      </RichTextEditorComponent>
    );
  }
}

export default App;

```

Customize placeholder style in React Rich text editor component

By using `e-rte-placeholder` class, you can customize the placeholder style.

[Class-component]

APP.JSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  placeholder = "Type something";
  render() {
    return (<RichTextEditorComponent placeholder={this.placeholder}>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>);
  }
}
export default App;

```

APP.TSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private placeholder: string = "Type something";
  public render() {

```

```

    return (
      <RichTextEditorComponent placeholder={this.placeholder}>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
      </RichTextEditorComponent>
    );
  }
}
export default App;

```

[Functional-component]

APP.JSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  let placeholder = "Type something";
  return (<RichTextEditorComponent placeholder={placeholder}>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  let placeholder: string = "Type something";
  return (
    <RichTextEditorComponent placeholder={placeholder}>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
  );
}
export default App;

```

Customize shortcut keys in React Rich text editor component

It can be achieved by using [formatter](#) property. We need to create `customformatterModel` to configure the `keyConfig` using `IHtmlFormatterModel` class and assign the same to the `formatter` property. Here, `ctrl+q` is configured to open the `Insert Hyperlink` dialog.

[Class-component]

APP.JSX

```

import { Count, HtmlEditor, HTMLFormatter, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';

```

```

class App extends React.Component {
  customHTMLModel = {
    // formatter is used to configure the custom key
    keyConfig: {
      'insert-link': 'ctrl+q', // confite the desired key
    }
  };
  formatter = new HTMLFormatter(this.customHTMLModel);
  // to configure custom key
  render() {
    return (<RichTextEditorComponent formatter={this.formatter}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]}/>
    </RichTextEditorComponent>);
  }
}
export default App;

```

APP.TSX

```

import { Count, HtmlEditor, HTMLFormatter, IHtmlFormatterModel, Image,
Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
'@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  public customHTMLModel: IHtmlFormatterModel = {
    // formatter is used to configure the custom key
    keyConfig: {
      'insert-link': 'ctrl+q', // confite the desired key
    }
  };
  public formatter: any = new HTMLFormatter(this.customHTMLModel);
  // to configure custom key
  public render() {
    return (
      <RichTextEditorComponent formatter={this.formatter}>

```

```

    <p>The Rich Text Editor component is WYSIWYG ("what you see is what
    you get") editor that provides the best user experience to create and update
    the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]} />
  </RichTextEditorComponent>
);
}
}
export default App;

```

[Functional-component]

APP.JSX

```

import { Count, HtmlEditor, HTMLFormatter, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let customHTMLModel = {
    // formatter is used to configure the custom key
    keyConfig: {
      'insert-link': 'ctrl+q', // confite the desired key
    }
  };
  let formatter = new HTMLFormatter(customHTMLModel);
  // to configure custom key
  return (<RichTextEditorComponent formatter={formatter}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
    get") editor that provides the best user experience to create and update the
    content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>

```

```

        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary functionality
on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]}/>
  </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

import { Count, HtmlEditor, HTMLFormatter, IHtmlFormatterModel, Image,
Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
'@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  let customHTMLModel: IHtmlFormatterModel = {
    // formatter is used to configure the custom key
    keyConfig: {
      'insert-link': 'ctrl+q', // confite the desired key
    }
  };
  let formatter: any = new HTMLFormatter(customHTMLModel);
  // to configure custom key
  return (
    <RichTextEditorComponent formatter={formatter}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
        <li>
          <p>Contains a modular library to load the necessary functionality
on demand.</p>
        </li>
        <li>
          <p>Provides a fully customizable toolbar.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Count, Image, Link, HtmlEditor,
QuickToolbar]}/>
    </RichTextEditorComponent>

```

```
);
}
export default App;
```

We need to import `IHtmlFormatterModel` and `HTMLFormatter` to configure the shortcut key.

Set the cursor at the specific range in React Rich text editor component

This can be achieved by using `setRange` method in the RTE using `NodeSelection` instance. In this below sample, we have passed the text node (specific location in RTE content) in `setStart` method and passed the range in `setRange` method of RTE.

[Class-component]

APP.JSX

```
{% raw %}
import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  onclick() {
    const element =
this.rteObj.contentModule.getDocument().getElementById("key");
    const selectioncursor = new NodeSelection();
    const range = document.createRange();
    range.setStart(element, 1); // to set the range
    selectioncursor.setRange(document, range); // to set the cursor
  }
  render() {
    return (<div>
      <RichTextEditorComponent ref={ (richtexteditor) => { this.rteObj =
richtexteditor; }}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p id="key"><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>
        </RichTextEditorComponent>
        <button id='btn' className="e-control e-btn" onClick={this.onclick =
this.onclick.bind(this)}> Set cursor position</button>
      </div>);
  }
}
export default App;
{% endraw %}
```

APP.TSX


```
{% raw %}
import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{},> {
  public rteObj: RichTextEditorComponent;
  public onclick() {
    const element: Element= (this.rteObj as
any).contentModule.getDocument().getElementById("key");
    const selectioncursor: NodeSelection = new NodeSelection();
    const range: Range = document.createRange();
    range.setStart(element, 1); // to set the range
    selectioncursor.setRange(document, range); // to set the cursor
  }
  public render() {
    return (
      <div>
        <RichTextEditorComponent ref={ (richtexteditor) => { this.rteObj =
richtexteditor! } }>
          <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
            Users can format their content using standard toolbar commands.</p>
          <p id="key"><b>Key features:</b></p>
          <ul>
            <li>
              <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
          </ul>
          <Inject services={ [Toolbar, Image, Link, HtmlEditor, QuickToolbar] }
/>
        </RichTextEditorComponent>
        <button id='btn' className="e-control e-btn" onClick={this.onclick=
this.onclick.bind(this)}> Set cursor position</button>
      </div>
    );
  }
}
export default App;
{% endraw %}
```

[Functional-component]

APP.JSX

```
{% raw %}
import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let rteObj;
  function onclick() {
    const element =
rteObj.contentModule.getDocument().getElementById("key");
  }
}
```

```

    const selectioncursor = new NodeSelection();
    const range = document.createRange();
    range.setStart(element, 1); // to set the range
    selectioncursor.setRange(document, range); // to set the cursor
  }
  return (<div>
    <RichTextEditorComponent ref={ (richtexteditor) => { rteObj =
richtexteditor; }}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
      Users can format their content using standard toolbar commands.</p>
      <p id="key"><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>
    </RichTextEditorComponent>
    <button id='btn' className="e-control e-btn" onClick={onClick =
onClick.bind(this)}> Set cursor position</button>
  </div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
import { HtmlEditor, Image, Inject, Link, NodeSelection, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
function App() {
  let rteObj: RichTextEditorComponent;
  function onclick() {
    const element: Element = (rteObj as
any).contentModule.getDocument().getElementById("key");
    const selectioncursor: NodeSelection = new NodeSelection();
    const range: Range = document.createRange();
    range.setStart(element, 1); // to set the range
    selectioncursor.setRange(document, range); // to set the cursor
  }
  return (
    <div>
      <RichTextEditorComponent ref={ (richtexteditor) => { rteObj =
richtexteditor! }}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
        Users can format their content using standard toolbar commands.</p>
        <p id="key"><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>

```

```

        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
    <button id='btn' className="e-control e-btn" onClick={onclick=
onclick.bind(this)}> Set cursor position</button>
  </div>
);
}
export default App;
{% endraw %}

```

Update value in React Rich text editor component

To achieve this, we need to bind the **keydown** event to the RTE content and capture the **ctrl + s** key press using its keyCode.

In the **keydown** event handler, the **updateValue** method is called to update the **value** property and then we can save the content in the required database using the same.

[Class-component]

APP.JSX

```

{% raw %}
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component {
  rteObj;
  created() {
    const instance = this.rteObj;
    this.rteObj.contentModule.getDocument().addEventListener("keydown",
(e) => {
      if (e.key === 's' && e.ctrlKey === true) {
        e.preventDefault(); // to prevent default ctrl+s action
        instance.updateValue(); // to update the value after editing
        // const value: any= instance.value; // you can get the RTE
        content to save in the desired database
      }
    });
  }
  render() {
    return (<RichTextEditorComponent ref={(richtexteditor) => {
this.rteObj = richtexteditor; }} created={this.created =
this.created.bind(this)}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>

```

```

        <p>Capable of handling markdown editing.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
  </RichTextEditorComponent>);
}
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-
richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  public rteObj: RichTextEditorComponent;
  public created(): void {
    const instance = this.rteObj;
    (this.rteObj as
any).contentModule.getDocument().addEventListener("keydown", (e: any)=>{
      if(e.key === 's' && e.ctrlKey===true){
        e.preventDefault(); // to prevent default ctrl+s action
        instance.updateValue(); // to update the value after editing
        // const value: any= instance.value; // you can get the RTE
        content to save in the desired database
      }
    });
  }
  public render() {
    return (
      <RichTextEditorComponent ref={(richtexteditor) => { this.rteObj =
richtexteditor! }} created={this.created = this.created.bind(this)} >
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
/>
      </RichTextEditorComponent>
    );
  }
}
export default App;
{% endraw %}

```

[functional-component]**APP.JSX**

```
{% raw %}
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  let rteObj;
  function created() {
    const instance = rteObj;
    rteObj.contentModule.getDocument().addEventListener("keydown", (e) =>
    {
      if (e.key === 's' && e.ctrlKey === true) {
        e.preventDefault(); // to prevent default ctrl+s action
        instance.updateValue(); // to update the value after editing
        // const value: any= instance.value; // you can get the RTE
        content to save in the desired database
      }
    });
  }
  return (<RichTextEditorComponent ref={(richtexteditor) => { rteObj =
richtexteditor; }} created={created.bind(this)}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}/>
  </RichTextEditorComponent>);
}
export default App;
{% endraw %}
```

APP.TSX

```
{% raw %}
import { HtmlEditor, Image, Inject, Link, QuickToolbar,
RichTextEditorComponent, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  let rteObj: RichTextEditorComponent;
  function created(): void {
    const instance = rteObj;
  }
}
```

```

    (rteObj as
any).contentModule.getDocument().addEventListener("keydown", (e: any) => {
    if (e.key === 's' && e.ctrlKey === true) {
        e.preventDefault(); // to prevent default ctrl+s action
        instance.updateValue(); // to update the value after editing
        // const value: any = instance.value; // you can get the RTE
        content to save in the desired database
    }
});
}
return (
    <RichTextEditorComponent ref={ (richtexteditor) => { rteObj =
richtexteditor! }} created={created.bind(this)} >
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
            <li>
                <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
            </li>
            <li>
                <p>Capable of handling markdown editing.</p>
            </li>
        </ul>
        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
    </RichTextEditorComponent>
);
}
export default App;
{% endraw %}

```

Rename images in server in React Rich text editor component

By using the [insertImageSettings](#) property, you can specify the server handler to upload the selected image. Then you can bind the [imageUploadSuccess](#) event, to receive the modified file name from the server and update it in the Rich Text Editor's insert image dialog.

[Class-component]

`ts

```
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
'@syncfusion/ej2-react-richtexteditor';
```

```
import * as React from 'react';
```

```
export class App extends React.Component<{}, {}> {
```

```
    private toolbarSettings: object = {
```

```
        items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
```

```
        'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
```

```
        'LowerCase', 'UpperCase', '|',
```

```
        'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
```

```

'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
};
private insertImageSettings: object = {
  saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/Rename",
  path: "../Images/"
};
public onImageUploadSuccess = (args: any) => {
  if (args.e.currentTarget.getResponseHeader('name') != null) {
    args.file.name = args.e.currentTarget.getResponseHeader('name');
    let filename: any = document.querySelectorAll(".e-file-name")[0];
    filename.innerHTML = args.file.name.replace(document.querySelectorAll(".e-file-type")[0].innerHTML,
    "");
    filename.title = args.file.name;
  }
}
public render() {
  return (
    <RichTextEditorComponent toolbarSettings={this.toolbarSettings}
    insertImageSettings={this.insertImageSettings}
    imageUploadSuccess={this.onImageUploadSuccess.bind(this)} >
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides
    the best user experience to create and update the content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
    <li>
    <p>Provides <IFRAME> and <DIV> modes</p>
    </li>
    <li>
    <p>Capable of handling markdown editing.</p>
    </li>
    <li>

```

```

<p>Contains a modular library to load the necessary functionality on demand.</p>
</li>
<li>
<p>Provides a fully customizable toolbar.</p>
</li>
</ul>
<Inject services={[Toolbar, Count, Image, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
}
`

```

[Functional-component]

```

`ts
import { Count, HtmlEditor, Image, Inject, Link, QuickToolbar, RichTextEditorComponent, Toolbar } from
 '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
const toolbarSettings: object = {
items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
};
const insertImageSettings: object = {
saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/Rename",
path: "../Images/"
};
function onImageUploadSuccess(args: any) {
if (args.e.currentTarget.getResponseHeader('name') != null) {

```



```

args.file.name = args.e.currentTarget.getResponseHeader('name');
let filename: any = document.querySelectorAll(".e-file-name")[0];
filename.innerHTML = args.file.name.replace(document.querySelectorAll(".e-file-type")[0].innerHTML,
"");
filename.title = args.file.name;
}
}
return (
<RichTextEditorComponent toolbarSettings={toolbarSettings} insertImageSettings={insertImageSettings}
imageUploadSuccess={onImageUploadSuccess.bind(this)} >
<p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides
the best user experience to create and update the content.
Users can format their content using standard toolbar commands.</p>
<p><b>Key features:</b></p>
<ul>
<li>
<p>Provides <IFRAME> and <DIV> modes</p>
</li>
<li>
<p>Capable of handling markdown editing.</p>
</li>
<li>
<p>Contains a modular library to load the necessary functionality on demand.</p>
</li>
<li>
<p>Provides a fully customizable toolbar.</p>
</li>
</ul>
<Inject services={[Toolbar, Count, Image, Link, HtmlEditor, QuickToolbar]} />
</RichTextEditorComponent>
);
}
export default App;
`

```

To configure the server-side handler, refer the below code.

```
`csharp
[AcceptVerbs("Post")]
public void Rename()
{
    try
    {
        var httpPostedFile = System.Web.HttpContext.Current.Request.Files["UploadFiles"];
        imageFile = httpPostedFile.FileName;
        if (httpPostedFile != null)
        {
            var fileSave = System.Web.HttpContext.Current.Server.MapPath("~/Images");
            if (!Directory.Exists(fileSave))
            {
                Directory.CreateDirectory(fileSave);
            }
            var fileName = Path.GetFileName(httpPostedFile.FileName);
            var fileSavePath = Path.Combine(fileSave, fileName);
            while (System.IO.File.Exists(fileSavePath))
            {
                imageFile = "rtelImage" + x + "-" + fileName;
                fileSavePath = Path.Combine(fileSave, imageFile);
                x++;
            }
            if (!System.IO.File.Exists(fileSavePath))
            {
                httpPostedFile.SaveAs(fileSavePath);
                HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
                Response.Clear();
                Response.Headers.Add("name", imageFile);
                Response.ContentType = "application/json; charset=utf-8";
                Response.StatusDescription = "File uploaded succesfully";
                Response.End();
            }
        }
    }
}
```

```

}
}
}
catch (Exception e)
{
    HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
    Response.Clear();
    Response.ContentType = "application/json; charset=utf-8";
    Response.StatusCode = 204;
    Response.Status = "204 No Content";
    Response.StatusDescription = e.Message;
    Response.End();
}
}
,

```

File attachment in React Rich text editor component

The Rich Text Editor allows you to attach a file based on the file upload. You can attach your files using the file upload or drag-and-drop from your local path. When the file upload gets success, the attachment link inserts into the content.

In the below sample, configure the saveUrl and path properties to achieve file attachments.

1.saveUrl: Provides service URL to save the files. 2.path: Specifies the location to store the image.

The following sample illustrates how to attach a file in the Rich Text Editor.

[Class-component]

```

`html
import * as ReactDOM from 'react-dom';
import * as React from 'react';

import { RichTextEditorComponent, HtmlEditor, Inject, Toolbar, QuickToolbar, Image, Link,
NodeSelection, IHtmlFormatterModel } from '@syncfusion/ej2-react-richtexteditor';
import { SampleBase } from '../common/sample-base';

import { UploaderComponent } from '@syncfusion/ej2-react-inputs/src/uploader';

export class ImageSample extends SampleBase<{}, {}> {
    public rteObj: RichTextEditorComponent;
    public uploadObj: UploaderComponent;
    public selection: NodeSelection = new NodeSelection();

```

```

public range: Range;
public asyncSettings: object;
public insertImageSetting: object;
public saveSelection: NodeSelection;
customHTMLModel: IHtmlFormatterModel;
public dropElement;
constructor(props: {}) {
  this.asyncSettings = {
    saveUrl: '[SERVICEHOSTEDPATH]/api/uploadbox/Save'
  };
  this.insertImageSetting = {
    saveUrl: '[SERVICEHOSTEDPATH]/api/uploadbox/Save',
    path: '../Files/'
  };
  this.dropElement = '.e-richtexteditor'
}
public onUploadSuccess(args: any): void {
  (this.rteObj.contentModule.getEditPanel() as HTMLElement).focus();
  this.range = this.selection.getRange(document);
  this.saveSelection = this.selection.save(this.range, document);
  var fileUrl = document.URL + this.rteObj.insertImageSettings.path + args.file.name;
  if (this.rteObj.formatter.getUndoRedoStack().length === 0) {
    this.rteObj.formatter.saveData();
  }
  saveSelection.restore();
  this.rteObj.executeCommand('createLink', { url: fileUrl, text: fileUrl, selection: saveSelection });
  this.rteObj.formatter.saveData();
  this.rteObj.formatter.enableUndo(rteObj);
  this.uploadObj.clearAll();
}
render() {
  return (
    <div className='control-pane'>

```

```

<div className='control-section' id="rteTools">
  <div className='rte-control-section'>
    <RichTextEditorComponent id="defaultRTE" ref={{richtexteditor} => { this.rteObj = richtexteditor; }}
    insertImageSettings={this.insertImageSetting}>
    <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
  </RichTextEditorComponent>
  <UploaderComponent id='fileUpload' type='file' ref={{scope} => { this.uploadObj = scope }}
  asyncSettings={this.asyncSettings} dropArea={this.dropElement}
  success={this.onUploadSuccess.bind(this)}
</UploaderComponent>
</div>
</div>
</div>
);
}
}
`

```

[Functional-component]

```

`html
import * as ReactDOM from 'react-dom';
import * as React from 'react';

import { RichTextEditorComponent, HtmlEditor, Inject, Toolbar, QuickToolbar, Image, Link,
NodeSelection, IHtmlFormatterModel } from '@syncfusion/ej2-react-richtexteditor';
import { SampleBase } from '../common/sample-base';
import { UploaderComponent } from '@syncfusion/ej2-react-inputs/src/uploader';

function ImageSample(){
  let rteObj: RichTextEditorComponent;
  let uploadObj: UploaderComponent;
  let selection: NodeSelection = new NodeSelection();
  let range: Range;
  let asyncSettings: object;
  let insertImageSetting: object;
  let saveSelection: NodeSelection;
  customHTMLModel: IHtmlFormatterModel;

```

```

let dropElement;
asyncSettings = {
  saveUrl: '[SERVICEHOSTEDPATH]/api/uploadbox/Save'
};
insertImageSetting = {
  saveUrl: '[SERVICEHOSTEDPATH]/api/uploadbox/Save',
  path: '../Files/'
};
dropElement = '.e-richtexteditor'
function onUploadSuccess(args: any): void {
  (rteObj.contentModule.getEditPanel() as HTMLElement).focus();
  range = selection.getRange(document);
  saveSelection = selection.save(range, document);
  var fileUrl = document.URL + rteObj.insertImageSettings.path + args.file.name;
  if (rteObj.formatter.getUndoRedoStack().length === 0) {
    rteObj.formatter.saveData();
  }
  saveSelection.restore();
  rteObj.executeCommand('createLink', { url: fileUrl, text: fileUrl, selection: saveSelection });
  rteObj.formatter.saveData();
  rteObj.formatter.enableUndo(rteObj);
  uploadObj.clearAll();
}
return (
  <div className='control-pane'>
    <div className='control-section' id="rteTools">
      <div className='rte-control-section'>
        <RichTextEditorComponent id="defaultRTE" ref={{(richtexteditor) => { rteObj = richtexteditor; }}}
          insertImageSettings={insertImageSetting}>
          <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]} />
        </RichTextEditorComponent>
        <UploaderComponent id='fileUpload' type='file' ref={{(scope) => { uploadObj = scope }}}
          asyncSettings={asyncSettings} dropArea={dropElement} success={onUploadSuccess.bind(this)}

```

```
</UploaderComponent>
</div>
</div>
</div>
);
}
export default ImageSample;
`
```

To configure server-side handler, refer the below code.

```
` csharp
int x = 0;
string file;
[AcceptVerbs("Post")]
public void Save()
{
    try
    {
        var httpPostedFile = System.Web.HttpContext.Current.Request.Files["UploadFiles"];
        file = httpPostedFile.FileName;
        if (httpPostedFile != null)
        {
            Console.WriteLine(System.Web.HttpContext.Current.Server.MapPath("~/Files"));
            var fileSave = System.Web.HttpContext.Current.Server.MapPath("~/Files");
            if (!Directory.Exists(fileSave))
            {
                Directory.CreateDirectory(fileSave);
            }
            var fileName = Path.GetFileName(httpPostedFile.FileName);
            var fileSavePath = Path.Combine(fileSave, fileName);
            while (System.IO.File.Exists(fileSavePath))
            {
                file = "rte" + x + "-" + fileName;
                fileSavePath = Path.Combine(fileSave, file);
            }
        }
    }
}
```

```

x++;
}
if (!System.IO.File.Exists(fileSavePath))
{
    httpPostedFile.SaveAs(fileSavePath);
    HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
    Response.Clear();
    Response.Headers.Add("name", file);
    Response.ContentType = "application/json; charset=utf-8";
    Response.StatusDescription = "File uploaded succesfully";
    Response.Headers.Add("url", fileSavePath);
    Response.End();
}
}
}
catch (Exception e)
{
    HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
    Response.Clear();
    Response.ContentType = "application/json; charset=utf-8";
    Response.StatusCode = 204;
    Response.Status = "204 No Content";
    Response.StatusDescription = e.Message;
    Response.End();
}
}
,

```

[Format code block in React Rich text editor component](#)

You can configure code block formatting as a separate toolbar button by adding the **InsertCode** keyword within the [toolbarSettings](#) items property.

The InsertCode button has a toggle state to apply code block formatting to the editor and remove code block formatting from the editor.

The following sample demonstrates how to config the InsertCode button in toolbar and set the background color to “pre” tag for highlighting the code block.

[Class-component]**APP.JSX**

```
import { HtmlEditor, Inject, Link, Image, RichTextEditorComponent,
QuickToolbar, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component {
  tools = {
    items: ['InsertCode']
  };
  render() {
    return (<RichTextEditorComponent toolbarSettings={this.tools}>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul>
        <li>
          <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
        </li>
        <li>
          <p>Capable of handling markdown editing.</p>
        </li>
      </ul>
      <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}>
    </RichTextEditorComponent>);
  }
}
export default App;
```

APP.TSX

```
import { HtmlEditor, Inject, Link, Image, RichTextEditorComponent,
QuickToolbar, Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
class App extends React.Component<{}, {}> {
  private tools: object = {
    items: ['InsertCode']
  }
  public render() {
    return (
      <RichTextEditorComponent toolbarSettings={this.tools}>
        <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar commands.</p>
        <p><b>Key features:</b></p>
        <ul>
          <li>
            <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
          </li>
          <li>
            <p>Capable of handling markdown editing.</p>
          </li>
        </ul>
      </RichTextEditorComponent>);
  }
}
```

```

        <Inject services={[Toolbar, Image, Link, HtmlEditor, QuickToolbar]}
    />
    </RichTextEditorComponent>
  );
}
}
export default App;

```

[Functional-component]

APP.JSX

```

import { HtmlEditor, Inject, Link, RichTextEditorComponent, QuickToolbar,
Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  const tools = {
    items: ['InsertCode']
  };
  return (<RichTextEditorComponent toolbarSettings={tools}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
    </ul>
    <Inject services={[Toolbar, Link, HtmlEditor, QuickToolbar]}>
    </RichTextEditorComponent>);
}
export default App;

```

APP.TSX

```

import { HtmlEditor, Inject, Link, RichTextEditorComponent, QuickToolbar,
Toolbar } from '@syncfusion/ej2-react-richtexteditor';
import * as React from 'react';
function App() {
  const tools = {
    items: ['InsertCode']
  };
  return (<RichTextEditorComponent toolbarSettings={tools}>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is what you
get") editor that provides the best user experience to create and update the
content.
    Users can format their content using standard toolbar commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>

```

```

        </li>
        <li>
            <p>Capable of handling markdown editing.</p>
        </li>
    </ul>
    <Inject services={[Toolbar, Link, HtmlEditor, QuickToolbar]}/>
</RichTextEditorComponent>);
}
export default App;

```

Schedule

Getting Started

Getting Started

This section briefly explains how to create [Link to the Video](#) component and configure its available functionalities in React environment, using Essential JS 2 [quickstart](#) seed repository.

To get start quickly with React Scheduler using the Create React App, you can check on this video:

Dependencies

The following list of dependencies are required to use the Scheduler component in your application.

```

`ts
|-- @syncfusion/ej2-react-schedule
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-schedule
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-excel-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-split-buttons
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-dropdowns
`

```

Installation and configuration

Setup for local development

To set-up a React application, choose any of the following ways. The best and easiest way is to use the [create-react-app](#). It sets up your development environment in JavaScript and improve your application for production. Refer to the [installation instructions](#) of `create-react-app`.

```
`bash
```

```
npx create-react-app my-app
```

```
cd my-app
```

```
npm start
```

```
,
```

or

```
`bash
```

```
yarn create react-app my-app
```

```
cd my-app
```

```
yarn start
```

```
,
```

To set-up a React application in `TypeScript` environment, run the following command.

```
`bash
```

```
npx create-react-app my-app --template typescript
```

```
cd my-app
```

```
npm start
```

```
,
```

Besides using the [npm](#) package runner tool, also create an application from the `npm init`. To begin with the `npm init`, upgrade the `npm` version to `npm 6+`.

```
`bash
```

```
npm init react-app my-app
```

```
cd my-app
```

```
npm start
```

```
,
```

Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](#) public registry.

To install Scheduler component, use the following command

```
,
```

```
npm install @syncfusion/ej2-react-schedule --save
```

```
,
```

Adding CSS reference

Add scheduler component's styles as given below in `src/App.css`.

```

`
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-react-schedule/styles/material.css";
`

```

To refer `App.css` in the application then import it in the `src/App.tsx` file.

In case, if you want to make use of the combined CSS files of entire components, then you can avail it from the root folder of Essential JS 2 package and reference it with the code shown below.

```

`css
@import '../..../node_modules/@syncfusion/ej2/material.css';
`

```

Module injection

Each view types available in scheduler are maintained as individual modules and to work with those views, it is necessary to inject the required modules. The following modules are available in scheduler namely,

- `Day` - Inject this module to work with the day view.
- `Week` - Inject this module to work with the week view.
- `WorkWeek` - Inject this module to work with the work week view.
- `Month` - Inject this module to work with the month view.
- `Agenda` - Inject this module to work with the agenda view.
- `MonthAgenda` - Inject this module for displaying month agenda view.
- `TimelineViews` - Inject this module to work with the timeline day, timeline week, timeline work week view.
- `TimelineMonth` - Inject this module to work with the timeline month view.

These modules should be injected into the schedule using the `Inject` method within the `app.tsx` file as shown below. On doing so, only the injected views will be loaded and displayed on the schedule.

[src/app/app.tsx]

`ts

```
<Inject services={[Day, Week, WorkWeek, Month, Agenda, MonthAgenda, TimelineViews, TimelineMonth]} />
```

,

Initialize the schedule

Import the Scheduler component to your `app.tsx` file using following code.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
const App = () => {
  return (<ScheduleComponent>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("schedule"));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
const App = () => {
  return (
    <ScheduleComponent>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  );
};
export default App;
ReactDOM.render(<App />, document.getElementById("schedule"));
```

Now, run the application in the browser using the following command.

,

npm start

,

Above demo will display the empty scheduler.

Populating appointments

- To populate the empty Scheduler with appointments, bind the event data to it by assigning the `dataSource` property either with valid JSON data or else with remote URL, from where the data will be fetched.

Here, the local JSON data is assigned to Scheduler's dataSource.

[src/app/app.tsx]

```
`ts
import * as React from 'react';
import {
  ScheduleComponent } from '@syncfusion/ej2-react-schedule';
const App = () => {
  const data: object [] = [
    {
      Id: 1,
      Subject: 'Meeting - 1',
      StartTime: new Date(2018, 1, 15, 10, 0),
      EndTime: new Date(2018, 1, 16, 12, 30),
      IsAllDay: false
    },
  ];
  const eventSettings = { dataSource: data }
  return (
    <ScheduleComponent height='550px' selectedDate= {new Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    </ScheduleComponent>
  );
}
export default App;`
```

You can also provide different names to these default fields, for which the custom names of those fields must be mapped appropriately within fields property as shown below.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
const App = () => {
  const data = [
    {
      Id: 2,
      Subject: 'Meeting',
```

```

        StartTime: new Date(2018, 1, 15, 10, 0),
        EndTime: new Date(2018, 1, 15, 12, 30),
        IsAllDay: false,
        Status: 'Completed',
        Priority: 'High'
    },
];
const fieldsData = {
    id: 'Id',
    subject: { name: 'Subject' },
    isAllDay: { name: 'IsAllDay' },
    startTime: { name: 'StartTime' },
    endTime: { name: 'EndTime' }
}
const eventSettings = { dataSource: data, fields: fieldsData }
return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
</ScheduleComponent>);
}
export default App;
ReactDOM.render(<App />, document.getElementById("schedule"));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject
} from '@syncfusion/ej2-react-schedule';
const App = () => {
    const data: object[] = [
        {
            Id: 2,
            Subject: 'Meeting',
            StartTime: new Date(2018, 1, 15, 10, 0),
            EndTime: new Date(2018, 1, 15, 12, 30),
            IsAllDay: false,
            Status: 'Completed',
            Priority: 'High'
        },
    ],
];
const fieldsData = {
    id: 'Id',
    subject: { name: 'Subject' },
    isAllDay: { name: 'IsAllDay' },
    startTime: { name: 'StartTime' },
    endTime: { name: 'EndTime' }
}
const eventSettings = { dataSource: data, fields: fieldsData }
return (
    <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings} >
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
);

```



```
}  
export default App;  
ReactDOM.render(<App />, document.getElementById("schedule"));
```

INDEX.HTML

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <title>Syncfusion React Schedule</title>  
  <meta charset="utf-8" />  
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
  <meta name="description" content="Essential JS 2 for React Components" />  
  <meta name="author" content="Syncfusion" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-buttons/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-calendars/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-dropdowns/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-inputs/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-navigations/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-popups/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-schedule/styles/material.css" rel="stylesheet" />  
  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />  
  <script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></script>  
  <script src="systemjs.config.js"></script>  
  <style>  
    #loader {  
      color: #008cff;  
      height: 40px;  
      left: 45%;  
      position: absolute;  
      top: 45%;  
      width: 30%;  
    }  
  </style>  
</head>  
<body>  
  <div id='schedule'>  
    <div id='loader'>Loading....</div>  
  </div>  
</body>  
</html>
```

The other fields available in Scheduler can be referred from [here](#).

Setting date

Scheduler usually displays the system date as its current date. To change the current date of Scheduler with specific date, define the `selectedDate` property.

[src/app/app.tsx]

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { defaultData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: defaultData }
  return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>);
};
export default App;
ReactDOM.render(<App />, document.getElementById("schedule"));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { defaultData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: defaultData }
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  )
};
export default App;
ReactDOM.render(<App />, document.getElementById("schedule"));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Setting view

Scheduler displays **week** view by default. To change the current view, define the applicable view name to the **currentView** property. The applicable view names are,

- Day
- Week
- WorkWeek
- Month
- Year
- Agenda
- MonthAgenda
- TimelineDay
- TimelineWeek
- TimelineWorkWeek

- TimelineMonth
- TimelineYear

```
`ts
import * as React from 'react';
import {
  ScheduleComponent, Day, Week, WorkWeek, Agenda, Month, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import './App.css';
const App = () => {
  return (
    <ScheduleComponent width='100%' height='550px' currentView='Month'
      selectedDate={new Date(2017, 11, 15)}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
        <ViewDirective option='Agenda' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, Agenda, Month]} />
    </ScheduleComponent>
  )
};
export default App;
```

Individual view customization

Each individual Scheduler views can be customized with its own options such as setting different start and end hour on Week and Work Week views, whereas hiding the weekend days on Month view alone. This can be achieved by defining views property to accept the array of object type, where each object depicts the individual view customization.

The output will display the Scheduler with the specified view configuration.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
```

```
import { ScheduleComponent, WorkWeek, Week, Month, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { defaultData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: defaultData }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='WorkWeek' startHour='10:00' endHour='18:00' />
      <ViewDirective option='Week' startHour='07:00' endHour='15:00' />
      <ViewDirective option='Month' showWeekend={false} />
    </ViewsDirective>
    <Inject services={[WorkWeek, Week, Month]} />
  </ScheduleComponent>);
}
export default App;
ReactDOM.render(<App />, document.getElementById("schedule"));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, WorkWeek, Week, Month, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { defaultData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: defaultData }
  return (
    <ScheduleComponent
      width='100%' height='550px' selectedDate={new Date(2018, 1, 15)}
      eventSettings={eventSettings}
    >
      <ViewsDirective>
        <ViewDirective option='WorkWeek' startHour='10:00' endHour='18:00' />
        <ViewDirective option='Week' startHour='07:00' endHour='15:00' />
        <ViewDirective option='Month' showWeekend={false} />
      </ViewsDirective>
      <Inject services={[WorkWeek, Week, Month]} />
    </ScheduleComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById("schedule"));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

You can also explore our [React Scheduler example](#) that shows how to use the toolbar buttons to play with Scheduler functionalities.

Creating a Next.js Application Using Syncfusion React Components

This section provides a step-by-step guide for setting up a Next.js application and integrating the Syncfusion React Schedule component.

What is Next.js?

[Next.js](#) is a React framework that makes it easy to build fast, SEO-friendly, and user-friendly web applications. It provides features such as server-side rendering, automatic code splitting, routing, and API routes, making it an excellent choice for building modern web applications.

Prerequisites

Before getting started with the Next.js application, ensure the following prerequisites are met:

- [Node.js 16.8](#) or later.
- The application is compatible with macOS, Windows, and Linux operating systems.

Create a Next.js application

To create a new Next.js application, use one of the commands that are specific to either NPM or Yarn.

NPM

```
npx create-next-app@latest
```

YARN

```
yarn create next-app
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: Users can specify the name of the project directly. Let's specify the name of the project as `ej2-nextjs-schedule`.

CMD

```
√ What is your project named? » ej2-nextjs-schedule
```

2. Select the required packages.

CMD

```
√ What is your project named? ... ej2-nextjs-schedule
√ Would you like to use TypeScript? ... No / `Yes`
√ Would you like to use ESLint? ... No / `Yes`
√ Would you like to use Tailwind CSS? ... `No` / Yes
√ Would you like to use `src/` directory? ... No / `Yes`
√ Would you like to use App Router? (recommended) ... No / `Yes`
√ Would you like to customize the default import alias? ... `No` / Yes
Creating a new Next.js app in D:\ej2-nextjs-schedule.
```

3. Once complete the above mentioned steps to create `ej2-nextjs-schedule`, navigate to the directory using the below command:

CMD

```
cd ej2-nextjs-schedule
```

The application is ready to run with default settings. Now, let's add Syncfusion components to the project.

Install Syncfusion React packages

Syncfusion React component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion React components in the project, install the corresponding npm package.

Here, the [React Schedule component](#) is used in the project. To install the React Schedule component, use the following command:

NPM

```
npm install @syncfusion/ej2-react-schedule --save
```

YARN

```
yarn add @syncfusion/ej2-react-schedule
```

Import Syncfusion CSS styles

Syncfusion React components come with [built-in themes](#), which are available in the installed packages. It's easy to adapt the Syncfusion React components to match the style of your application by referring to one of the built-in themes.

Import the **Material** theme into the **src/app/globals.css** file and removed the existing styles in that file, as shown below:

GLOBALS.CSS

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-react-schedule/styles/material.css";
```

To know more about built-in themes and CSS reference for individual components, refer to the [themes](#) section.

Add Syncfusion React component

Follow the below steps to add the React Schedule component to the Next.js project:

1. Before adding the Schedule component to your markup, create a **datasource.tsx** file within the **src/app/** folder and add the Schedule component data.

DATASOURCE.TSX

```
export let timelineResourceData: Object[] = [
{
  Id: 61,
  Subject: 'Decoding',
  StartTime: new Date(2018, 3, 4, 9, 30),
  EndTime: new Date(2018, 3, 4, 10, 30),
  IsAllDay: false,
  ProjectId: 2,
  TaskId: 2
}, {
```



```
Id: 62,
Subject: 'Bug Automation',
StartTime: new Date(2018, 3, 4, 13, 30),
EndTime: new Date(2018, 3, 4, 16, 30),
IsAllDay: false,
ProjectId: 2,
TaskId: 1
}, {
Id: 63,
Subject: 'Functionality testing',
StartTime: new Date(2018, 3, 4, 9),
EndTime: new Date(2018, 3, 4, 10, 30),
IsAllDay: false,
ProjectId: 1,
TaskId: 1
}, {
Id: 64,
Subject: 'Resolution-based testing',
StartTime: new Date(2018, 3, 4, 12),
EndTime: new Date(2018, 3, 4, 13),
IsAllDay: false,
ProjectId: 1,
TaskId: 1
}, {
Id: 65,
Subject: 'Test report Validation',
StartTime: new Date(2018, 3, 4, 15),
EndTime: new Date(2018, 3, 4, 18),
IsAllDay: false,
ProjectId: 1,
TaskId: 1
}, {
Id: 66,
Subject: 'Test case correction',
StartTime: new Date(2018, 3, 4, 14),
EndTime: new Date(2018, 3, 4, 16),
IsAllDay: false,
ProjectId: 1,
TaskId: 2
}, {
Id: 67,
Subject: 'Bug fixing',
StartTime: new Date(2018, 3, 4, 14, 30),
EndTime: new Date(2018, 3, 4, 18, 30),
IsAllDay: false,
ProjectId: 2,
TaskId: 2
}, {
Id: 68,
Subject: 'Run test cases',
StartTime: new Date(2018, 3, 4, 17, 30),
EndTime: new Date(2018, 3, 4, 19, 30),
IsAllDay: false,
ProjectId: 1,
TaskId: 2
}, {
Id: 70,
```

```

Subject: 'Bug Automation',
StartTime: new Date(2018, 3, 4, 18, 30),
EndTime: new Date(2018, 3, 4, 20),
IsAllDay: false,
ProjectId: 2,
TaskId: 1
}
];

```

2. Then, import and define the Schedule component in the **src/app/page.tsx** file, as shown below:

PAGE.TSX

```

'use client'
import {
  Week, Month, Agenda, ScheduleComponent, ViewsDirective, ViewDirective,
  EventSettingsModel, ResourcesDirective, ResourceDirective, Inject, Resize,
  DragAndDrop
} from '@syncfusion/ej2-react-schedule';
import { timelineResourceData } from './datasource';
export default function Home() {
  const eventSettings: EventSettingsModel = { dataSource: timelineResourceData
  }
  const group = { byGroupID: false, resources: ['Projects', 'Categories'] }
  const projectData: Object[] = [
    { text: 'PROJECT 1', id: 1, color: '#cb6bb2' },
    { text: 'PROJECT 2', id: 2, color: '#56ca85' },
    { text: 'PROJECT 3', id: 3, color: '#df5286' },
  ];
  const categoryData: Object[] = [
    { text: 'Development', id: 1, color: '#1aaa55' },
    { text: 'Testing', id: 2, color: '#7fa900' }
  ];
  return (
    <>
    <h2>Syncfusion React Schedule Component</h2>
    <ScheduleComponent width='100%' height='550px' currentView='Month'
    selectedDate={new Date(2018, 3, 4)} eventSettings={eventSettings}
    group={group} >
    <ViewsDirective>
    <ViewDirective option='Week' />
    <ViewDirective option='Month' />
    <ViewDirective option='Agenda' />
    </ViewsDirective>
    <ResourcesDirective>
    <ResourceDirective field='ProjectId' title='Choose Project' name='Projects'
    allowMultiple={false}
    dataSource={projectData} textField='text' idField='id' colorField='color'>
    </ResourceDirective>
    <ResourceDirective field='TaskId' title='Category' name='Categories'
    allowMultiple={true}
    dataSource={categoryData} textField='text' idField='id' colorField='color'>
    </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[Week, Month, Agenda, Resize, DragAndDrop]} />
    </ScheduleComponent>
  )
}

```

```
</>
)
}
```

Run the application

To run the application, use the following command:

NPM

```
npm run dev
```

YARN

```
yarn run dev
```

To learn more about the functionality of the Schedule component, refer to the [documentation](#).

[View the NEXT.js Schedule sample in the GitHub repository.](#)

Getting Started with the React Schedule Component in the Preact Framework

This article provides a step-by-step guide for setting up a [Preact](#) project and integrating the Syncfusion React Schedule component.

Preact is a fast and lightweight JavaScript library for building user interfaces. It's often used as an alternative to larger frameworks like React. The key difference is that Preact is designed to be smaller in size and faster in performance, making it a good choice for projects where file size and load times are critical factors.

Prerequisites

[System requirements for Syncfusion React UI components](#)

Set up the Preact project

To create a new **Preact** project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm init preact
```

```
`
```

or

```
`bash
```

```
yarn init preact
```

```
`
```

Using one of the above commands will lead you to set up additional configurations for the project, as below:

1\ Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

T Preact - Fast 3kB alternative to React with the same modern API

```
|
```

- Project directory:

```
| my-project
```

```
—
```

```
`
```

2\ Choose JavaScript as the framework variant to build this Preact project using JavaScript and React.

```
`bash
```

T Preact - Fast 3kB alternative to React with the same modern API

```
|
```

- Project language:

```
| > JavaScript
```

```
| TypeScript
```

```
—
```

```
`
```

3\ Then configure the project as below for this article.

```
`bash
```

T Preact - Fast 3kB alternative to React with the same modern API

```
|
```

- Use router?

```
| Yes / > No
```

```
—
```

```
|
```

- Prerender app (SSG)?

```
| Yes / > No
```

```
—
```

```
|
```

- Use ESLint?

```
| Yes / > No
```

```
—
```

5\ Upon completing the aforementioned steps to create `my-project`, run the following command to jump into the project directory:

```
`bash
cd my-project
`
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

Add the Syncfusion React packages

Syncfusion React component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion React components in the project, install the corresponding npm package.

This article uses the [React Schedule component](#) as an example. To use the React Schedule component in the project, the `@syncfusion/ej2-react-schedule` package needs to be installed using the following command:

```
`bash
npm install @syncfusion/ej2-react-schedule --save
`
```

or

```
`bash
yarn add @syncfusion/ej2-react-schedule
`
```

Import Syncfusion CSS styles

You can import themes for the Syncfusion React component in various ways, such as using CSS or SASS styles from npm packages, CDN, CRG and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to theme's in a React project.

In this article, the `Material 3` theme is applied using CSS styles, which are available in installed packages. The necessary `Material 3` CSS styles for the Schedule component and its dependents were imported into the `src/style.css` file.

~/SRC/STYLE.CSS

```
@import "../node_modules/@syncfusion/ej2-base/styles/material3.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material3.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material3.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material3.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material3.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material3.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material3.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material3.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material3.css";
@import "../node_modules/@syncfusion/ej2-react-
schedule/styles/material3.css";
```

The order of importing CSS styles should be in line with its dependency graph.

Add the Syncfusion React component

Follow the below steps to add the React Schedule component to the Vite project:

1\ Before adding the Schedule component to your markup, import the Schedule component in the **src/index.jsx** file.

~/SRC/INDEX.JSX

```
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
```

2\ Then, define the Schedule component in the **src/index.jsx** file, as shown below:

~/SRC/INDEX.JSX

```
import { render } from 'preact';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import './style.css';
export function App() {
  return (
    <ScheduleComponent>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  );
}
render(<App />, document.getElementById('app'));
```

Run the project

To run the project, use the following command:

``bash`

`npm run dev`

`,`

or

``bash`

`yarn run dev`

`,`

The output will appear as follows:



See also

[Getting Started with the Syncfusion React UI Component](#)

Module injection in React Schedule component

The crucial step on creating a Scheduler with required views, is to import and inject the required modules. The modules that are available on Scheduler to work with its related functionalities are as follows.

- **Day** - Inject this module to work with day view.
- **Week** - Inject this module to work with week view.
- **WorkWeek** - Inject this module to work with work week view.
- **Month** - Inject this module to work with month view.
- **Year** - Inject this module to work with year view.
- **Agenda** - Inject this module to work with agenda view.
- **MonthAgenda** - Inject this module to work with month agenda view.
- **TimelineViews** - Inject this module to work with timeline day, timeline week, and timeline work week views.
- **TimelineMonth** - Inject this module to work with timeline month view.
- **TimelineYear** - Inject this module to work with timeline year view.
- **DragAndDrop** - Inject this module to allow drag and drop of appointments on Scheduler.
- **Resize** - Inject this module for enabling the resize functionality of appointments on Scheduler.

Module injection

The required modules should be injected into the Scheduler using the **Inject** method of Schedule within the **app.ts** file as shown below. On doing so, only the injected module functionalities will be loaded and can be worked with Scheduler.

[src/app.tsx]

```
`ts
```

```
<Inject services={[Day, Week, WorkWeek, Month, Agenda, MonthAgenda ]} />
```

Note: If a Scheduler `currentView` is set to any one of the available views without injecting that respective view module, then a script error will occur and the Scheduler will not render.

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Scheduler interactions in React Schedule component

The following table describes the Scheduler actions and also illustrates how those actions are carried out through mouse and touch interactions on Scheduler.

Actions	Mouse interaction	Touch interaction
-----	-----	-----
Single click or tap on cells	Single click on a cell to select a cell.	Single tapping on cells, will display a + icon on the cell. Tapping on it again will open the new event editor window.
Multiple cell selection	Single click on a cell and drag the selection to other cells to enable multiple cell selection.	No multiple cell selection is allowed using touch gestures.
Event selection	Single click on an event to select it.	Tap holding on events, select an event and opens a small popup at the top holding the options to edit or delete. The popup also displays the selected event's subject.
Multiple event selection and deletion	Pressing <code>Ctrl</code> key and altogether single clicking on multiple events one after the other will enable multiple event selection. Pressing <code>Delete</code> key after event selection will delete all the selected events.	Tap hold an event to select it, which opens a small popup at the top holding the options to edit or delete. As a continuation of this action, keep on single tapping on other events, to enable multiple event selection. Also, the popup displayed at the top remains in opened state, showing the count of the number of selected events. Pressing <code>Delete</code> option from the popup will delete all the selected events.
Date navigation	Clicking on the previous or next date navigation icons in the header bar allows you to navigate between dates.	Swiping the scheduler view port to the left or right will allow you to navigate between the dates on touch devices. You can prevent the swiping action by disabling <code>allowSwiping</code> property. NOTE: Swiping does not work when horizontal scroller present in the Scheduler. You can also make use of the previous and next navigation icons at the header bar to navigate.
View navigation	Click on an event and try moving it over the Scheduler to enable drag and drop action.	The view options are available within the popup options at the top right extreme end of the header bar and you can choose the view from it.
Drag and drop	Click on an event and try moving it over the Scheduler to enable drag and drop action.	Tap hold the event and try moving it over the Scheduler to enable drag and drop action.
Event resizing	Hover the mouse across the extremities or edges of the Scheduler events and when the mouse pointer changes into resize handler, now click and start resizing an event to the desired time range.	Touch the event extremities and start resizing the events directly.
Tooltip	Hover the mouse pointer over the events or resource header and the tooltip will be displayed.	Tap holding the events will open the tooltip on events.

| Open editor window | Double click on cells or events to open the editor window. | Double click on cells or events to open the editor window. Single tap on cells, which displays a + icon on the cell. Now, tap on it again to open the new event editor window. To open the editor on events, single tap on it and then click on the edit icon to open the editor window in **Edit** mode. |

| Open quick info popup | Single clicking on a cell will open a quick popup prompting for new event creation. Single clicking on an event will open a popup displaying event information along with the option to edit and delete it. | No quick info popup is available while single tapping on cells. Single tapping on events, opens the popup showing event information. |

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler exampleLink to the Video](#) to know how to present and manipulate data.

Appointments in React Schedule component

Appointments can be anything that are scheduled for a specific time period. It can be created on varied time range and each appointments are categorized based on this range. The Scheduler events can be categorized as,

- Normal events
- Spanned events
- All-day events
- Recurring events

To have a quick glance on how to add appointments to the React Scheduler and some of its advanced event-handling options, watch this video:

Normal events

Represents an appointment that is created for any specific time interval within a day.

Creating a normal event

The following example depicts how to define a event on the Scheduler, with event data being loaded from simple data.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
const App = () => {
  const data = [{
    Id: 1,
    Subject: 'Paris',
    StartTime: new Date(2023, 1, 15, 10, 0),
    EndTime: new Date(2023, 1, 15, 12, 30),
  }];
  const eventSettings = { dataSource: data }
  return <ScheduleComponent height='550px' selectedDate={new Date(2023, 1,
15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>;
}
```

```
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const data: object[] = [{
    Id: 1,
    Subject: 'Paris',
    StartTime: new Date(2023, 1, 15, 10, 0),
    EndTime: new Date(2023, 1, 15, 12, 30),
  }];
  const eventSettings: EventSettingsModel = { dataSource: data }
  return <ScheduleComponent height='550px' selectedDate={new Date(2023, 1,
15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
```

```

<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Spanned events

Represents an appointment that is created for more than 24 hours, and usually displayed on the all-day row. Also, represents another type of appointment that is created for more than one day but less than 24 hours, and usually displayed appropriately on both the days.

For example, if an appointment is created for two days say from November 25, 2018 – 11.00 PM to November 26, 2018 2.00 AM but less than 24 hours time interval, then the appointment is split into two partitions and will be displayed on both the days.

All-day events

Represents an appointment that is created for an entire day such as holiday events. It is usually displayed separately in an all-day row, a separate row for all-day appointments below the date header section. In Timeline views, the all-day appointments displays in the working space area, and no separate all-day row is present in that view.

To change normal appointment into all-day event, set `isAllDay` field to true.

Hide all-day row events

You can make use of the CSS customization to prevent the display of all-day row appointments on the Scheduler UI.

`ts

```

<style>
.e-schedule .e-date-header-wrap .e-schedule-table thead {
display: none;
}
</style>
`

```

Expand all day appointments view on initial load

When you have larger number of appointments in all-day view, you can show all all-day events using **dataBound** event on at initial load. So, user don't have to click the toggle to expand all-day events.

INDEX.JSX

```

import { useRef, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
Resize, DragAndDrop } from '@syncfusion/ej2-react-schedule';
const App = () => {
  const initialLoad = useRef(true);
  const scheduleRef = useRef(null);
  useEffect(() => {
    if (initialLoad.current) {
      const allDayAppointmentSection =
scheduleRef.current.element.querySelector('.e-all-day-appointment-section');
      if (allDayAppointmentSection) {
        allDayAppointmentSection.click();
      }
      initialLoad.current = false;
    }
  }, []);
  const data = [
    {
      EndTime: new Date(2022, 3, 30, 0, 0),
      Id: '2',
      IsAllDay: true,
      StartTime: new Date(2022, 3, 29, 0, 0),
      Subject: 'Plumbing Checklist | Jaimungal | 3671 :: Pool',
    },
    {
      EndTime: new Date(2022, 3, 30, 0, 0),
      Id: '4',
      IsAllDay: true,
      StartTime: new Date(2022, 3, 28, 0, 0),
      Subject: 'Underground Plumbing | Jaimungal | 3671 :: Pool',
    },
    {
      EndTime: new Date(2022, 3, 30, 12, 30),
      Id: '7',
      IsAllDay: true,
      StartTime: new Date(2022, 3, 24, 0, 0),
      Subject: 'Steel/ Checklist | VP Highland Model | 3719 :: Pool',
    },
  ],

```

```

    {
      EndTime: new Date(2022, 3, 30, 0, 0),
      Id: '9',
      IsAllDay: true,
      StartTime: new Date(2022, 3, 29, 0, 0),
      Subject: 'Tile Selections/ Pavers/ Finish | VP Highland Model | 3719 ::
Pool',
    },
    {
      EndTime: new Date(2022, 3, 30, 0, 0),
      Id: '10',
      IsAllDay: true,
      StartTime: new Date(2022, 3, 26, 0, 0),
      Subject: 'Layout/ Form Rebar Shell | VP Highland Model | 3719 :: Pool',
    },
    {
      EndTime: new Date(2022, 3, 30, 0, 0),
      Id: '10',
      IsAllDay: true,
      StartTime: new Date(2022, 3, 26, 0, 0),
      Subject: ' VP Highland Model | 3719 :: Pool',
    },
  ];
  const eventSettings = { dataSource: data };
  return (
    <ScheduleComponent
      height="550px"
      ref={scheduleRef}
      selectedDate={new Date(2022, 3, 26)}
      eventSettings={eventSettings}
    >
      <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
Resize]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
Resize, DragAndDrop, EventSettingsModel } from '@syncfusion/ej2-react-
schedule';
const App = () => {
  const initialLoad = useRef<Boolean>(true);
  const scheduleRef = useRef<ScheduleComponent>(null);
  useEffect(() => {
    if (initialLoad.current) {
      const allDayAppointmentSection =
scheduleRef.current.element.querySelector('.e-all-day-appointment-section');
      if (allDayAppointmentSection) {
        allDayAppointmentSection.click();
      }
    }
  }, []);
};

```

```

    }
    initialLoad.current = false;
  }
}, []);
const data: object[] = [
  {
    EndTime: new Date(2022, 3, 30, 0, 0),
    Id: '2',
    IsAllDay: true,
    StartTime: new Date(2022, 3, 29, 0, 0),
    Subject: 'Plumbing Checklist | Jaimungal | 3671 :: Pool',
  },
  {
    EndTime: new Date(2022, 3, 30, 0, 0),
    Id: '4',
    IsAllDay: true,
    StartTime: new Date(2022, 3, 28, 0, 0),
    Subject: 'Underground Plumbing | Jaimungal | 3671 :: Pool',
  },
  {
    EndTime: new Date(2022, 3, 30, 12, 30),
    Id: '7',
    IsAllDay: true,
    StartTime: new Date(2022, 3, 24, 0, 0),
    Subject: 'Steel/ Checklist | VP Highland Model | 3719 :: Pool',
  },
  {
    EndTime: new Date(2022, 3, 30, 0, 0),
    Id: '9',
    IsAllDay: true,
    StartTime: new Date(2022, 3, 29, 0, 0),
    Subject: 'Tile Selections/ Pavers/ Finish | VP Highland Model | 3719 ::
Pool',
  },
  {
    EndTime: new Date(2022, 3, 30, 0, 0),
    Id: '10',
    IsAllDay: true,
    StartTime: new Date(2022, 3, 26, 0, 0),
    Subject: 'Layout/ Form Rebar Shell | VP Highland Model | 3719 :: Pool',
  },
  {
    EndTime: new Date(2022, 3, 30, 0, 0),
    Id: '10',
    IsAllDay: true,
    StartTime: new Date(2022, 3, 26, 0, 0),
    Subject: ' VP Highland Model | 3719 :: Pool',
  },
],
];
const eventSettings: EventSettingsModel = { dataSource: data };
return (
  <ScheduleComponent
    height="550px"
    ref={scheduleRef}
    selectedDate={new Date(2022, 3, 26)}
    eventSettings={eventSettings}
  >

```

```

    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
    Resize]} />
  </ScheduleComponent>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008c8f;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,

```

```

        .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Customize the rendering of the spanned events

By default, Scheduler will renders the spanned events (appointment with more than 24 hours duration) in the all-day row by setting `AllDayRow` will the default type renders to the `spannedEventPlacement` option within the `eventSettings` property. Now we can customize rendering of the that events inside the work cells itself by modifying the `spannedEventPlacement` option as `TimeSlot`. In this following example, shows how to render the spanned appointments inside the work cells as follows.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
const App = () => {
    const data = [{
        Id: 1,
        Subject: 'Paris',
        StartTime: new Date(2018, 1, 15, 10, 0),
        EndTime: new Date(2018, 1, 17, 12, 30),
        IsAllDay: false
    }, {
        Id: 2,
        Subject: 'London',
        StartTime: new Date(2018, 1, 16, 12, 0),
        EndTime: new Date(2018, 1, 18, 13, 0),
        IsAllDay: false
    }
    ];
    const eventSettings = { dataSource: data, spannedEventPlacement:
'TimeSlot' };
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';

```



```
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const data: object[] = [{
    Id: 1,
    Subject: 'Paris',
    StartTime: new Date(2018, 1, 15, 10, 0),
    EndTime: new Date(2018, 1, 17, 12, 30),
    IsAllDay: false
  }, {
    Id: 2,
    Subject: 'London',
    StartTime: new Date(2018, 1, 16, 12, 0),
    EndTime: new Date(2018, 1, 18, 13, 0),
    IsAllDay: false
  }];
  const eventSettings: EventSettingsModel = { dataSource: data,
    spannedEventPlacement: 'TimeSlot' };
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
```

```

<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Recurring events

Represents an appointment that is created for a certain time interval and occurring repeatedly on a daily, weekly, monthly or yearly basis at the same time interval based on the provided recurrence rule. Usually, the recurring events are indicated by a repeat marker added at the bottom-right position.

Creating a recurring event

The following example depicts how to create a recurring event on Scheduler with the specific recurrence rule. In the following example, an event is made to repeat on daily mode and ends after 5 occurrences.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
const App = () => {
    const data = [{
        Id: 2,
        Subject: 'Paris',
        StartTime: new Date(2018, 1, 15, 10, 0),

```

```

        EndTime: new Date(2018, 1, 15, 12, 30),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=5',
    }];
    const eventSettings = { dataSource: data };
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
    const data: object[] = [{
        Id: 2,
        Subject: 'Paris',
        StartTime: new Date(2018, 1, 15, 10, 0),
        EndTime: new Date(2018, 1, 15, 12, 30),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=5',
    }];
    const eventSettings: EventSettingsModel = { dataSource: data };
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008c9f;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Adding exceptions

A few instance of the recurrence series can be excluded on specific dates, by adding those exceptional dates to the `recurrenceException` field. These date values should be given in the ISO date time format with no hyphens(-) separating the date elements.

For example, 22nd February 2018 can be represented as 20180222. Also, the time part being represented in UTC format needs to add "Z" after the time portion with no space. "07:30:00 UTC" is therefore represented as "073000Z".

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
const App = () => {
  const data = [{
    Id: 1,
    Subject: 'Paris',
    StartTime: new Date(2018, 0, 28, 10, 0),
    EndTime: new Date(2018, 0, 28, 12, 30),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=8',
    RecurrenceException: '20180129T043000Z,20180131T043000Z,20180202T043000Z'
  }];
  const eventSettings = { dataSource: data };
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 0,
28)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>;
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const data: object[] = [{
    Id: 1,
    Subject: 'Paris',
    StartTime: new Date(2018, 0, 28, 10, 0),
    EndTime: new Date(2018, 0, 28, 12, 30),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=8',
    RecurrenceException: '20180129T043000Z,20180131T043000Z,20180202T043000Z'
  }];
  const eventSettings: EventSettingsModel = { dataSource: data };
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 0,
28)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
```

```

<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
      background: green;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>

```

```
</html>
```

Editing an occurrence from a series

To dynamically edit a particular occurrence from an event series and display it on the initial load of Scheduler, the edited occurrence needs to be added as a new event to the dataSource collection, with an additional `recurrenceID` field defined to it. The `recurrenceID` field of edited occurrence usually maps the ID value of the parent event.

In this example, a recurring instance that displays on the date 30th Jan 2018 is edited with different timings. Therefore, this particular date is excluded from the parent recurring event that repeats from 28th January 2018 to 4th February 2018. This can be done by adding the `recurrenceException` field with the excluded date value on the parent event. Also, the edited occurrence event which is created as a new event should carry the `recurrenceID` field pointing to the parent event's `Id` value.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
const App = () => {
  const data = [{
    Id: 1,
    Subject: 'Scrum Meeting',
    StartTime: new Date(2018, 0, 28, 10, 0),
    EndTime: new Date(2018, 0, 28, 12, 30),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=8',
    RecurrenceException: '20180130T043000Z'
  },
  {
    Id: 2,
    Subject: 'Scrum Meeting',
    StartTime: new Date(2018, 0, 30, 9, 0),
    EndTime: new Date(2018, 0, 30, 10, 30),
    Description: "Meeting time changed based on team activities.",
    RecurrenceID: 1
  }
  ];
  const eventSettings = { dataSource: data };
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 0,
28)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
```

```

ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const data: object[] = [{
    Id: 1,
    Subject: 'Scrum Meeting',
    StartTime: new Date(2018, 0, 28, 10, 0),
    EndTime: new Date(2018, 0, 28, 12, 30),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=8',
    RecurrenceException: '20180130T043000Z'
  },
  {
    Id: 2,
    Subject: 'Scrum Meeting',
    StartTime: new Date(2018, 0, 30, 9, 0),
    EndTime: new Date(2018, 0, 30, 10, 30),
    Description: "Meeting time changed based on team activities.",
    RecurrenceID: 1
  }
  ];
  const eventSettings: EventSettingsModel = { dataSource: data,
editFollowingEvents: true };
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 0,
28)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
  };
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />

```



```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Edit only the current and following events

To edit only the current and following events enable the property `editFollowingEvents` within `eventSettings` property. The edited occurrence needs to be added as a new event to the `dataSource` collection, with an additional `followingID` field defined to it. The `followingID` field of edited occurrence usually maps the ID value of the immediate parent event.

In this example, a recurring instance that displays on the date 30th Jan 2018 and its following dates are edited with different subject. Therefore, this particular date and its following dates are excluded from the parent recurring event that repeats from 28th January 2018 to 4th February 2018. This can be done by updating the `recurrenceRule` field with the until date value on the parent event. Also, the edited events which is created as a new event should carry the `followingID` field pointing to the immediate parent event's `Id` value.

INDEX.JSX

```
import * as React from 'react';
```

```
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
const App = () => {
    const data = [{
        Id: 1,
        Subject: 'Scrum Meeting',
        StartTime: new Date(2018, 0, 28, 10, 0),
        EndTime: new Date(2018, 0, 28, 12, 30),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;UNTIL=20180129T043000Z;',
    },
    {
        Id: 2,
        Subject: 'Scrum Meeting - Following Edited',
        StartTime: new Date(2018, 0, 30, 10, 0),
        EndTime: new Date(2018, 0, 30, 12, 30),
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;UNTIL=20180204T043000Z;',
        FollowingID: 1
    }
    ];
    let eventSettings = { dataSource: data, editFollowingEvents: true };
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 0, 28)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]}/>
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, EventSettingsModel,
    Inject
} from '@syncfusion/ej2-react-schedule';
const App = () => {
    const data: object[] = [{
        Id: 1,
        Subject: 'Scrum Meeting',
        StartTime: new Date(2018, 0, 28, 10, 0),
        EndTime: new Date(2018, 0, 28, 12, 30),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;UNTIL=20180129T043000Z;',
    },
    {
        Id: 2,
        Subject: 'Scrum Meeting - Following Edited',
        StartTime: new Date(2018, 0, 30, 10, 0),
        EndTime: new Date(2018, 0, 30, 12, 30),
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;UNTIL=20180204T043000Z;',
        FollowingID: 1
    }
    ];
```

```

    let eventSettings: EventSettingsModel = { dataSource: data,
editFollowingEvents: true };
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 0,
28)} eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  };
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
  </style>

```

```

        .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
        .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Recurrence options and rules

Events can be repeated on a daily, weekly, monthly or yearly basis based on the recurrence rule which accepts the string value. The following details should be assigned to the `recurrenceRule` property to generate the recurring instances.

- Repeat type - daily/weekly/monthly/yearly.
- How many times it needs to be repeated?
- The interval duration.
- The time period to render the appointment, etc.

There are four repeat types available namely,

- **Daily** - Creates the recurring instances on daily basis.
- **Weekly** - Creates the recurring instances on weekly basis for the selected days.
- **Monthly** - Creates the recurring instances on monthly basis for the selected months and other provided recurrence criteria.
- **Yearly** - Creates the recurring instances on yearly basis.

Recurrence properties

The properties based on which the recurrence appointments are created with its respective time period are depicted in the following table. Also, the valid rule string can be referred from [iCalendar](#) specifications.

Refer [iCalendar](#) specifications for valid recurrence rule string.

Property	Purpose	Example
FREQ	Maintains the repeat type (Daily, Weekly, Monthly, Yearly) value of the appointment.	FREQ=DAILY;INTERVAL=1
INTERVAL	Maintains the interval value of the appointments. When you create the daily appointment at an interval of 2, the appointments are rendered on the days Monday, Wednesday and Friday (Creates an appointment on all days by leaving the interval of one day gap).	FREQ=DAILY;INTERVAL=2

| COUNT | It holds the appointment's count value. When the COUNT value is 10, then 10 instances of appointments are created in the recurrence series. | FREQ=DAILY;INTERVAL=1;COUNT=10|

| UNTIL | This property holds the end date value (in ISO format) denoting when the recurrence actually ends. | FREQ=DAILY;INTERVAL=1;UNTIL=20180530T041343Z;|

| BYDAY | It holds the day value(s), representing on which the appointments actually renders. Create the weekly appointment, and select the day(s) from the day options (Monday/Tuesday/Wednesday/Thursday/Friday/Saturday/Sunday). When Monday is selected, the first two letters of the selected day "MO" is saved in the BYDAY property. When multiple days are selected, the values are separated by commas. | FREQ=WEEKLY;INTERVAL=1;BYDAY=MO,WE;COUNT=10|

| BYMONTHDAY | This property is used to store the date value of the Month, while creating the Month recurrence appointment. When you create a Monthly recurrence appointment for every 3rd day of the month, then BYMONTHDAY holds the value 3 and creates the appointment on 3rd day of every month. | FREQ=MONTHLY;BYMONTHDAY=3;INTERVAL=1;COUNT=10|

| BYMONTH | This property is used to store the index value of the selected Month while creating the yearly appointments. When you create the yearly appointment on June month, the index value of June month 6 will get stored in the BYMONTH field. The appointment is created on every 6th month of a year. | FREQ=YEARLY;BYMONTHDAY=16;BYMONTH=6;INTERVAL=1;COUNT=10|

| BYSETPOS | This property is used to store the index value of the week. When you create the monthly appointment in second week of a month, the index value of the second week (2) is stored in BYSETPOS. | FREQ=MONTHLY;BYDAY=MO;BYSETPOS=2;COUNT=10|

The default recurrence related validation has been included for recurrence appointments similar to the one available in Outlook. The validation usually occurs during the recurrence appointment creation, editing, drag and drop or resizing of the recurrence appointments and also if any single occurrence changes.

Daily Frequency

| Description | Example |

|-----|-----|

| Daily recurring event that never ends | FREQ=DAILY; INTERVAL=1 |

| Daily recurring event that ends after 5 occurrences | FREQ=DAILY; INTERVAL=1; COUNT=5 |

| Daily recurring event that ends exactly on 12/12/2018 | FREQ=DAILY; INTERVAL=1; UNTIL=20181212T041343Z |

| Daily event that recurs on alternative days and repeats for 10 occurrences | FREQ=DAILY; INTERVAL=2; COUNT=10 |

| Daily recurring appointment ends on 12/12/2018 by excluding single occurrence on 12/10/2018 | FREQ=DAILY; INTERVAL=2; UNTIL=20181212T041343Z; |

Weekly Frequency

| Description | Example |

|-----|-----|

| Weekly recurring event that repeats on every Monday, Wednesday and Friday and never ends | FREQ=WEEKLY; INTERVAL=1; BYDAY=MO,WE,FR |

| Repeats every week Thursday and ends after 10 occurrences | FREQ=WEEKLY; INTERVAL=1; BYDAY=TH; COUNT=10 |

| Repeats every week Monday and ends on 12/12/2018 | FREQ=WEEKLY; INTERVAL=1; BYDAY=MO; UNTIL=20181212T041343Z |

| Repeats on Monday, Wednesday and Friday of alternative weeks and ends after 10 occurrences | FREQ=WEEKLY; INTERVAL=2; BYDAY=MO, WE, FR; COUNT=10 |

| Repeats every week on weekdays and ends after 12/12/2018 by excluding single occurrence on 12/10/2018 | FREQ=WEEKLY; BYDAY=MO, TU, WE, TH, FR; UNTIL=20181212T041343Z; |

Monthly Frequency

| Description | Example |

|-----|-----|

| Monthly recurring event that repeats on every 15th day of a month and never ends | FREQ=MONTHLY; BYMONTHDAY=15; INTERVAL=1 |

| Monthly recurring event that repeats on every 16th day of a month and ends after 10 occurrences | FREQ=MONTHLY; BYMONTHDAY=16; INTERVAL=1; COUNT=10 |

| Repeats every 17th day of a month and ends on 12/12/2018 | FREQ=MONTHLY; BYMONTHDAY=17; INTERVAL=1; UNTIL=20181212T041343Z |

| Repeats every 2nd Friday of a month and never ends | FREQ=MONTHLY; BYDAY=FR; BYSETPOS=2; INTERVAL=1 |

| Repeats every 4th Wednesday of a month and ends after 10 occurrences | FREQ=MONTHLY; BYDAY=WE; BYSETPOS=4; INTERVAL=1; COUNT=10 |

| Repeats every 4th Friday of a month and ends on 12/12/2018 | FREQ=MONTHLY; BYDAY=FR; BYSETPOS=4; INTERVAL=1; UNTIL=20181212T041343Z |

Yearly Frequency

| Description | Example |

|-----|-----|

| Yearly event that repeats on every 15th day of December month and never ends | FREQ=YEARLY; BYMONTHDAY=15; BYMONTH=12; INTERVAL=1 |

| Event that repeats on every 10th day of December month and ends after 10 occurrences | FREQ=YEARLY; BYMONTHDAY=10; BYMONTH=12; INTERVAL=1; COUNT=10 |

| Repeats on every 12th day of December month and ends on 12/12/2025 | FREQ=YEARLY; BYMONTHDAY=12; BYMONTH=12; INTERVAL=1; UNTIL=20251212T041343Z |

| Repeats on every 3rd Friday of December month and never ends | FREQ=YEARLY; BYDAY=FR; BYMONTH=12; BYSETPOS=3; INTERVAL=1 |

| Repeats on every 3rd Tuesday of December month and ends after 10 occurrences | FREQ=YEARLY; BYDAY=TU; BYMONTH=12; BYSETPOS=3; INTERVAL=1; COUNT=10 |

| Repeats on every 4th Wednesday of December month and ends on 12/12/2028 | FREQ=YEARLY; BYDAY=WE; BYMONTH=12; BYSETPOS=4; INTERVAL=1; UNTIL=20181212T041343Z |

Recurrence Validation

The built-in validation support has been added by default for recurring appointments during its creation, edit, drag and drop or resize action. The following are the possible validation alerts that displays on Scheduler while creating or editing the recurring events.

Validation messages	Description
---------------------	-------------

----- -----	
-------------	--

The recurrence pattern is not valid. This alert will raise, when the selected recurrence rule value is not a valid one. For example, when you try to select the end date value (using Until option) for a recurring event, which occurs before the start date, an alert will popup out saying that the chosen pattern is invalid.
--

The changes made to specific instances of this series will be cancelled and those events will match the series again. This alert will raise, when you try to edit the whole series, whose occurrence might have been already edited. For example, If there are five occurrences and one of the occurrence is already edited. Now, when you try to edit the entire series, you will get this validation alert.

The duration of the event must be shorter than how frequently it occurs. Shorten the duration, or change the recurrence pattern in the recurrence event editor. This validation will occur, if the event duration is longer than the selected frequency. For example, if you create a recurring appointment with two days duration in Daily frequency with no intervals set to it, you may get this alert.

Some months have fewer than the selected date. For these months, the occurrence will fall on the last date of the month. When you try to create a recurring appointment on 31st of every month, where few months won't have 31 days and in this scenario, you will get this alert.
--

Two occurrences of the same event cannot occur on the same day. This validation will occur, when you try to edit or move any single occurrence to some other date, where another occurrence of the same event is already present.

Event fields

The Scheduler dataSource usually holds the event instances, where each of the instance includes a collection of appropriate [fields](#). It is mandatory to map these fields with the equivalent fields of database, when remote data is bound to it. When the local JSON data is bound, then the field names defined within the instances needs to be mapped with the scheduler event fields correctly.

To create an event on Scheduler, it is enough to define the **startTime** and **endTime**. Also **id** field becomes mandatory to process CRUD actions on appropriate events.

Built-in fields

The built-in fields available on Scheduler event object are as follows.

Field name	Description
------------	-------------

----- -----	
-------------	--

id The id field needs to be defined as mandatory and this field usually assigns a unique ID value to each of the events.

subject The subject field is optional, and usually assigns the summary text to each of the events.

startTime The startTime field defines the start time of an event and it is mandatory to provide it for any of the valid event objects.

| **endTime** | The **endTime** field defines the end time of an event and it is mandatory to provide the end time for any of the valid event objects. |

| **startTimezone** | It maps the **startTimezone** field from the dataSource and usually accepts the valid IANA timezone names. It is assumed that the value provided for this field is taken into consideration while processing the **startTime** field. When this field is not mapped with any timezone names, then the events will be processed based on the timezone assigned to the Scheduler. |

| **endTimezone** | It maps the **endTimezone** field from the dataSource and usually accepts the valid IANA timezone names. It is assumed that the value provided for this field is taken into consideration while processing the **endTime** field. When this field is not mapped with any timezone names, then the events will be processed based on the timezone assigned to the Scheduler. |

| **location** | It maps the **location** field from the dataSource and the location text value will be displayed over the events. |

| **description** | It maps the **description** field from the dataSource and denotes the event description which is optional. |

| **isAllDay** | The **isAllDay** field is mapped from the dataSource and is used to denote whether an event is created for an entire day or for specific time alone. Usually, an event with **isAllDay** field set to true will be considered as an all-day event. |

| **recurrenceID** | It maps the **recurrenceID** field from dataSource and usually holds the ID value of the parent recurrence event. This field is applicable only for the edited occurrence events. |

| **recurrenceRule** | It maps the **recurrenceRule** field from dataSource and holds the recurrence rule value in a string format. Also, it uniquely identifies whether the event belongs to a recurring type or normal ones. |

| **recurrenceException** | It maps the **recurrenceException** field from dataSource and is used to hold the collection of exception dates, on which the recurring occurrences needs to be excluded. The **recurrenceException** should be specified in UTC format. |

| **isReadOnly** | It maps the **isReadOnly** field from dataSource. It is mainly used to make specific appointments as readonly when set to **true**. |

| **isBlock** | It maps the **isBlock** field from dataSource. It is used to block the particular time ranges in the Scheduler and prevents the event creation on those time slots. |

Binding different field names

When the fields of event instances has the default mapping name, it is not mandatory to map them manually. If a Scheduler's dataSource holds the events collection with different field names, then it is necessary to map them with its equivalent field name within the **eventSettings** property.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
const App = () => {
  const data = [{
    TravelId: 2,
    TravelSummary: 'Paris',
```



```

    DepartureTime: new Date(2018, 1, 15, 10, 0),
    ArrivalTime: new Date(2018, 1, 15, 12, 30),
    FullDay: false,
    Source: 'London',
    Comments: 'Summer vacation planned for outstation.',
    Origin: 'Asia/Yekaterinburg',
    Destination: 'Asia/Yekaterinburg'
  }];
  const fieldsData = {
    id: 'TravelId',
    subject: { name: 'TravelSummary' },
    isAllDay: { name: 'FullDay' },
    location: { name: 'Source' },
    description: { name: 'Comments' },
    startTime: { name: 'DepartureTime' },
    endTime: { name: 'ArrivalTime' },
    startTimezone: { name: 'Origin' },
    endTimezone: { name: 'Destination' }
  }
  const eventSettings = { dataSource: data, fields: fieldsData }
  return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const data: object[] = [{
    TravelId: 2,
    TravelSummary: 'Paris',
    DepartureTime: new Date(2018, 1, 15, 10, 0),
    ArrivalTime: new Date(2018, 1, 15, 12, 30),
    FullDay: false,
    Source: 'London',
    Comments: 'Summer vacation planned for outstation.',
    Origin: 'Asia/Yekaterinburg',
    Destination: 'Asia/Yekaterinburg'
  }];
  const fieldsData = {
    id: 'TravelId',
    subject: { name: 'TravelSummary' },
    isAllDay: { name: 'FullDay' },
    location: { name: 'Source' },
    description: { name: 'Comments' },
    startTime: { name: 'DepartureTime' },
    endTime: { name: 'ArrivalTime' },
  }

```

```

    startTimezone: { name: 'Origin' },
    endTimezone: { name: 'Destination' }
  }
  const eventSettings: EventSettingsModel = { dataSource: data, fields:
fieldsData }
  return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>

```

```

        .e-past-app {
            background-color: chocolate !important;
        }
        .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
        .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

The mapper field `id` is of string type and has no additional validation options, whereas all other fields are of `Object` type and has additional options.

Event field settings

Each field of the Scheduler events are provided with additional settings such as options to set default value, to map with appropriate data source fields, to validate every event fields and to provide label values for those fields in the event window.

Options	Description
default	Accepts the default value to the applicable fields (Subject, Location and Description), when no values are provided to them from dataSource.
name	Accepts the field name to be mapped from the dataSource fields.
title	Accepts the label values to be displayed for the fields of event editor.
validation	Defines the validation rules to be applied on the event fields within the event editor.

In following example, the Subject field in event editor will display its appropriate label as **Summary**. When no subject value is provided while saving an event, then the appointment will be saved with the default subject value as **Add Summary**.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
const App = () => {
    const data = [{
        TravelId: 2,
        TravelSummary: 'Paris',
        DepartureTime: new Date(2018, 1, 15, 10, 0),
        ArrivalTime: new Date(2018, 1, 15, 12, 30),
        Source: 'London',
    }];

```

```

        Comments: 'Summer vacation planned for outstation.'
    }];
    const fieldsData = {
        id: 'TravelId',
        subject: { name: 'TravelSummary', title: 'Summary', default: 'Add
Summary' },
        location: { name: 'Source', default: 'USA' },
        description: { name: 'Comments' },
        startTime: { name: 'DepartureTime' },
        endTime: { name: 'ArrivalTime' }
    }
    const eventSettings = { dataSource: data, fields: fieldsData }
    return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
    const data: object[] = [{
        TravelId: 2,
        TravelSummary: 'Paris',
        DepartureTime: new Date(2018, 1, 15, 10, 0),
        ArrivalTime: new Date(2018, 1, 15, 12, 30),
        Source: 'London',
        Comments: 'Summer vacation planned for outstation.'
    }];
    const fieldsData = {
        id: 'TravelId',
        subject: { name: 'TravelSummary', title: 'Summary', default: 'Add
Summary' },
        location: { name: 'Source', default: 'USA' },
        description: { name: 'Comments' },
        startTime: { name: 'DepartureTime' },
        endTime: { name: 'ArrivalTime' }
    }
    const eventSettings: EventSettingsModel = { dataSource: data, fields:
fieldsData }
    return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
      background: green;
    }
  </style>
</head>

```

```
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

Adding Custom fields

Apart from the default Scheduler fields, the user can include 'n' number of custom fields for appointments. The following code example shows how to include two custom fields namely **Status** and **Priority** within event collection. It is not necessary to bind the custom fields within the `eventSettings`. However, those additional fields can be accessed easily, for internal processing as well as from application end.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
const App = () => {
  const data = [{
    Id: 2,
    Subject: 'Meeting',
    StartTime: new Date(2018, 1, 15, 10, 0),
    EndTime: new Date(2018, 1, 15, 12, 30),
    IsAllDay: false,
    Status: 'Completed',
    Priority: 'High'
  }];
  const fieldsData = {
    id: 'Id',
    subject: { name: 'Subject' },
    isAllDay: { name: 'IsAllDay' },
    startTime: { name: 'StartTime' },
    endTime: { name: 'EndTime' }
  }
  const eventSettings = { dataSource: data, fields: fieldsData }
  return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
```

```

const App = () => {
  const data: Object[] = [{
    Id: 2,
    Subject: 'Meeting',
    StartTime: new Date(2018, 1, 15, 10, 0),
    EndTime: new Date(2018, 1, 15, 12, 30),
    IsAllDay: false,
    Status: 'Completed',
    Priority: 'High'
  }];
  const fieldsData = {
    id: 'Id',
    subject: { name: 'Subject' },
    isAllDay: { name: 'IsAllDay' },
    startTime: { name: 'StartTime' },
    endTime: { name: 'EndTime' }
  }
  const eventSettings: EventSettingsModel = { dataSource: data, fields:
fieldsData }
  return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />

```

```

<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Customize the order of the overlapping events

By default, the scheduler will render the overlapping events based on the start and end time. Now we can customize the order of the overlapping events based on the custom fields by using the `sortComparer` property grouped under the `eventSettings` property. The following code example shows how to sort the appointments based on the custom field as follows.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject
} from '@syncfusion/ej2-react-schedule';
const App = () => {
    const comparerFun = (args) => {
        args.sort((event1, event2) => event1.RankId.localeCompare(event2.RankId,
            undefined, { numeric: true }));
        return args;
    }
    const data = [{
        Id: 1,
        Subject: 'Rank 1',
    },
    {
        Id: 2,
        Subject: 'Rank 2',
    },
    {
        Id: 3,
        Subject: 'Rank 3',
    },
    {
        Id: 4,
        Subject: 'Rank 4',
    },
    {
        Id: 5,
        Subject: 'Rank 5',
    },
    {
        Id: 6,
        Subject: 'Rank 6',
    },
    {
        Id: 7,
        Subject: 'Rank 7',
    },
    {
        Id: 8,
        Subject: 'Rank 8',
    },
    {
        Id: 9,
        Subject: 'Rank 9',
    },
    {
        Id: 10,
        Subject: 'Rank 10',
    },
    {
        Id: 11,
        Subject: 'Rank 11',
    },
    {
        Id: 12,
        Subject: 'Rank 12',
    },
    {
        Id: 13,
        Subject: 'Rank 13',
    },
    {
        Id: 14,
        Subject: 'Rank 14',
    },
    {
        Id: 15,
        Subject: 'Rank 15',
    },
    {
        Id: 16,
        Subject: 'Rank 16',
    },
    {
        Id: 17,
        Subject: 'Rank 17',
    },
    {
        Id: 18,
        Subject: 'Rank 18',
    },
    {
        Id: 19,
        Subject: 'Rank 19',
    },
    {
        Id: 20,
        Subject: 'Rank 20',
    },
    {
        Id: 21,
        Subject: 'Rank 21',
    },
    {
        Id: 22,
        Subject: 'Rank 22',
    },
    {
        Id: 23,
        Subject: 'Rank 23',
    },
    {
        Id: 24,
        Subject: 'Rank 24',
    },
    {
        Id: 25,
        Subject: 'Rank 25',
    },
    {
        Id: 26,
        Subject: 'Rank 26',
    },
    {
        Id: 27,
        Subject: 'Rank 27',
    },
    {
        Id: 28,
        Subject: 'Rank 28',
    },
    {
        Id: 29,
        Subject: 'Rank 29',
    },
    {
        Id: 30,
        Subject: 'Rank 30',
    },
    {
        Id: 31,
        Subject: 'Rank 31',
    },
    {
        Id: 32,
        Subject: 'Rank 32',
    },
    {
        Id: 33,
        Subject: 'Rank 33',
    },
    {
        Id: 34,
        Subject: 'Rank 34',
    },
    {
        Id: 35,
        Subject: 'Rank 35',
    },
    {
        Id: 36,
        Subject: 'Rank 36',
    },
    {
        Id: 37,
        Subject: 'Rank 37',
    },
    {
        Id: 38,
        Subject: 'Rank 38',
    },
    {
        Id: 39,
        Subject: 'Rank 39',
    },
    {
        Id: 40,
        Subject: 'Rank 40',
    },
    {
        Id: 41,
        Subject: 'Rank 41',
    },
    {
        Id: 42,
        Subject: 'Rank 42',
    },
    {
        Id: 43,
        Subject: 'Rank 43',
    },
    {
        Id: 44,
        Subject: 'Rank 44',
    },
    {
        Id: 45,
        Subject: 'Rank 45',
    },
    {
        Id: 46,
        Subject: 'Rank 46',
    },
    {
        Id: 47,
        Subject: 'Rank 47',
    },
    {
        Id: 48,
        Subject: 'Rank 48',
    },
    {
        Id: 49,
        Subject: 'Rank 49',
    },
    {
        Id: 50,
        Subject: 'Rank 50',
    },
    {
        Id: 51,
        Subject: 'Rank 51',
    },
    {
        Id: 52,
        Subject: 'Rank 52',
    },
    {
        Id: 53,
        Subject: 'Rank 53',
    },
    {
        Id: 54,
        Subject: 'Rank 54',
    },
    {
        Id: 55,
        Subject: 'Rank 55',
    },
    {
        Id: 56,
        Subject: 'Rank 56',
    },
    {
        Id: 57,
        Subject: 'Rank 57',
    },
    {
        Id: 58,
        Subject: 'Rank 58',
    },
    {
        Id: 59,
        Subject: 'Rank 59',
    },
    {
        Id: 60,
        Subject: 'Rank 60',
    },
    {
        Id: 61,
        Subject: 'Rank 61',
    },
    {
        Id: 62,
        Subject: 'Rank 62',
    },
    {
        Id: 63,
        Subject: 'Rank 63',
    },
    {
        Id: 64,
        Subject: 'Rank 64',
    },
    {
        Id: 65,
        Subject: 'Rank 65',
    },
    {
        Id: 66,
        Subject: 'Rank 66',
    },
    {
        Id: 67,
        Subject: 'Rank 67',
    },
    {
        Id: 68,
        Subject: 'Rank 68',
    },
    {
        Id: 69,
        Subject: 'Rank 69',
    },
    {
        Id: 70,
        Subject: 'Rank 70',
    },
    {
        Id: 71,
        Subject: 'Rank 71',
    },
    {
        Id: 72,
        Subject: 'Rank 72',
    },
    {
        Id: 73,
        Subject: 'Rank 73',
    },
    {
        Id: 74,
        Subject: 'Rank 74',
    },
    {
        Id: 75,
        Subject: 'Rank 75',
    },
    {
        Id: 76,
        Subject: 'Rank 76',
    },
    {
        Id: 77,
        Subject: 'Rank 77',
    },
    {
        Id: 78,
        Subject: 'Rank 78',
    },
    {
        Id: 79,
        Subject: 'Rank 79',
    },
    {
        Id: 80,
        Subject: 'Rank 80',
    },
    {
        Id: 81,
        Subject: 'Rank 81',
    },
    {
        Id: 82,
        Subject: 'Rank 82',
    },
    {
        Id: 83,
        Subject: 'Rank 83',
    },
    {
        Id: 84,
        Subject: 'Rank 84',
    },
    {
        Id: 85,
        Subject: 'Rank 85',
    },
    {
        Id: 86,
        Subject: 'Rank 86',
    },
    {
        Id: 87,
        Subject: 'Rank 87',
    },
    {
        Id: 88,
        Subject: 'Rank 88',
    },
    {
        Id: 89,
        Subject: 'Rank 89',
    },
    {
        Id: 90,
        Subject: 'Rank 90',
    },
    {
        Id: 91,
        Subject: 'Rank 91',
    },
    {
        Id: 92,
        Subject: 'Rank 92',
    },
    {
        Id: 93,
        Subject: 'Rank 93',
    },
    {
        Id: 94,
        Subject: 'Rank 94',
    },
    {
        Id: 95,
        Subject: 'Rank 95',
    },
    {
        Id: 96,
        Subject: 'Rank 96',
    },
    {
        Id: 97,
        Subject: 'Rank 97',
    },
    {
        Id: 98,
        Subject: 'Rank 98',
    },
    {
        Id: 99,
        Subject: 'Rank 99',
    },
    {
        Id: 100,
        Subject: 'Rank 100',
    },
    {
        Id: 101,
        Subject: 'Rank 101',
    },
    {
        Id: 102,
        Subject: 'Rank 102',
    },
    {
        Id: 103,
        Subject: 'Rank 103',
    },
    {
        Id: 104,
        Subject: 'Rank 104',
    },
    {
        Id: 105,
        Subject: 'Rank 105',
    },
    {
        Id: 106,
        Subject: 'Rank 106',
    },
    {
        Id: 107,
        Subject: 'Rank 107',
    },
    {
        Id: 108,
        Subject: 'Rank 108',
    },
    {
        Id: 109,
        Subject: 'Rank 109',
    },
    {
        Id: 110,
        Subject: 'Rank 110',
    },
    {
        Id: 111,
        Subject: 'Rank 111',
    },
    {
        Id: 112,
        Subject: 'Rank 112',
    },
    {
        Id: 113,
        Subject: 'Rank 113',
    },
    {
        Id: 114,
        Subject: 'Rank 114',
    },
    {
        Id: 115,
        Subject: 'Rank 115',
    },
    {
        Id: 116,
        Subject: 'Rank 116',
    },
    {
        Id: 117,
        Subject: 'Rank 117',
    },
    {
        Id: 118,
        Subject: 'Rank 118',
    },
    {
        Id: 119,
        Subject: 'Rank 119',
    },
    {
        Id: 120,
        Subject: 'Rank 120',
    },
    {
        Id: 121,
        Subject: 'Rank 121',
    },
    {
        Id: 122,
        Subject: 'Rank 122',
    },
    {
        Id: 123,
        Subject: 'Rank 123',
    },
    {
        Id: 124,
        Subject: 'Rank 124',
    },
    {
        Id: 125,
        Subject: 'Rank 125',
    },
    {
        Id: 126,
        Subject: 'Rank 126',
    },
    {
        Id: 127,
        Subject: 'Rank 127',
    },
    {
        Id: 128,
        Subject: 'Rank 128',
    },
    {
        Id: 129,
        Subject: 'Rank 129',
    },
    {
        Id: 130,
        Subject: 'Rank 130',
    },
    {
        Id: 131,
        Subject: 'Rank 131',
    },
    {
        Id: 132,
        Subject: 'Rank 132',
    },
    {
        Id: 133,
        Subject: 'Rank 133',
    },
    {
        Id: 134,
        Subject: 'Rank 134',
    },
    {
        Id: 135,
        Subject: 'Rank 135',
    },
    {
        Id: 136,
        Subject: 'Rank 136',
    },
    {
        Id: 137,
        Subject: 'Rank 137',
    },
    {
        Id: 138,
        Subject: 'Rank 138',
    },
    {
        Id: 139,
        Subject: 'Rank 139',
    },
    {
        Id: 140,
        Subject: 'Rank 140',
    },
    {
        Id: 141,
        Subject: 'Rank 141',
    },
    {
        Id: 142,
        Subject: 'Rank 142',
    },
    {
        Id: 143,
        Subject: 'Rank 143',
    },
    {
        Id: 144,
        Subject: 'Rank 144',
    },
    {
        Id: 145,
        Subject: 'Rank 145',
    },
    {
        Id: 146,
        Subject: 'Rank 146',
    },
    {
        Id: 147,
        Subject: 'Rank 147',
    },
    {
        Id: 148,
        Subject: 'Rank 148',
    },
    {
        Id: 149,
        Subject: 'Rank 149',
    },
    {
        Id: 150,
        Subject: 'Rank 150',
    },
    {
        Id: 151,
        Subject: 'Rank 151',
    },
    {
        Id: 152,
        Subject: 'Rank 152',
    },
    {
        Id: 153,
        Subject: 'Rank 153',
    },
    {
        Id: 154,
        Subject: 'Rank 154',
    },
    {
        Id: 155,
        Subject: 'Rank 155',
    },
    {
        Id: 156,
        Subject: 'Rank 156',
    },
    {
        Id: 157,
        Subject: 'Rank 157',
    },
    {
        Id: 158,
        Subject: 'Rank 158',
    },
    {
        Id: 159,
        Subject: 'Rank 159',
    },
    {
        Id: 160,
        Subject: 'Rank 160',
    },
    {
        Id: 161,
        Subject: 'Rank 161',
    },
    {
        Id: 162,
        Subject: 'Rank 162',
    },
    {
        Id: 163,
        Subject: 'Rank 163',
    },
    {
        Id: 164,
        Subject: 'Rank 164',
    },
    {
        Id: 165,
        Subject: 'Rank 165',
    },
    {
        Id: 166,
        Subject: 'Rank 166',
    },
    {
        Id: 167,
        Subject: 'Rank 167',
    },
    {
        Id: 168,
        Subject: 'Rank 168',
    },
    {
        Id: 169,
        Subject: 'Rank 169',
    },
    {
        Id: 170,
        Subject: 'Rank 170',
    },
    {
        Id: 171,
        Subject: 'Rank 171',
    },
    {
        Id: 172,
        Subject: 'Rank 172',
    },
    {
        Id: 173,
        Subject: 'Rank 173',
    },
    {
        Id: 174,
        Subject: 'Rank 174',
    },
    {
        Id: 175,
        Subject: 'Rank 175',
    },
    {
        Id: 176,
        Subject: 'Rank 176',
    },
    {
        Id: 177,
        Subject: 'Rank 177',
    },
    {
        Id: 178,
        Subject: 'Rank 178',
    },
    {
        Id: 179,
        Subject: 'Rank 179',
    },
    {
        Id: 180,
        Subject: 'Rank 180',
    },
    {
        Id: 181,
        Subject: 'Rank 181',
    },
    {
        Id: 182,
        Subject: 'Rank 182',
    },
    {
        Id: 183,
        Subject: 'Rank 183',
    },
    {
        Id: 184,
        Subject: 'Rank 184',
    },
    {
        Id: 185,
        Subject: 'Rank 185',
    },
    {
        Id: 186,
        Subject: 'Rank 186',
    },
    {
        Id: 187,
        Subject: 'Rank 187',
    },
    {
        Id: 188,
        Subject: 'Rank 188',
    },
    {
        Id: 189,
        Subject: 'Rank 189',
    },
    {
        Id: 190,
        Subject: 'Rank 190',
    },
    {
        Id: 191,
        Subject: 'Rank 191',
    },
    {
        Id: 192,
        Subject: 'Rank 192',
    },
    {
        Id: 193,
        Subject: 'Rank 193',
    },
    {
        Id: 194,
        Subject: 'Rank 194',
    },
    {
        Id: 195,
        Subject: 'Rank 195',
    },
    {
        Id: 196,
        Subject: 'Rank 196',
    },
    {
        Id: 197,
        Subject: 'Rank 197',
    },
    {
        Id: 198,
        Subject: 'Rank 198',
    },
    {
        Id: 199,
        Subject: 'Rank 199',
    },
    {
        Id: 200,
        Subject: 'Rank 200',
    },
    {
        Id: 201,
        Subject: 'Rank 201',
    },
    {
        Id: 202,
        Subject: 'Rank 202',
    },
    {
        Id: 203,
        Subject: 'Rank 203',
    },
    {
        Id: 204,
        Subject: 'Rank 204',
    },
    {
        Id: 205,
        Subject: 'Rank 205',
    },
    {
        Id: 206,
        Subject: 'Rank 206',
    },
    {
        Id: 207,
        Subject: 'Rank 207',
    },
    {
        Id: 208,
        Subject: 'Rank 208',
    },
    {
        Id: 209,
        Subject: 'Rank 209',
    },
    {
        Id: 210,
        Subject: 'Rank 210',
    },
    {
        Id: 211,
        Subject: 'Rank 211',
    },
    {
        Id: 212,
        Subject: 'Rank 212',
    },
    {
        Id: 213,
        Subject: 'Rank 213',
    },
    {
        Id: 214,
        Subject: 'Rank 214',
    },
    {
        Id: 215,
        Subject: 'Rank 215',
    },
    {
        Id: 216,
        Subject: 'Rank 216',
    },
    {
        Id: 217,
        Subject: 'Rank 217',
    },
    {
        Id: 218,
        Subject: 'Rank 218',
    },
    {
        Id: 219,
        Subject: 'Rank 219',
    },
    {
        Id: 220,
        Subject: 'Rank 220',
    },
    {
        Id: 221,
        Subject: 'Rank 221',
    },
    {
        Id: 222,
        Subject: 'Rank 222',
    },
    {
        Id: 223,
        Subject: 'Rank 223',
    },
    {
        Id: 224,
        Subject: 'Rank 224',
    },
    {
        Id: 225,
        Subject: 'Rank 225',
    },
    {
        Id: 226,
        Subject: 'Rank 226',
    },
    {
        Id: 227,
        Subject: 'Rank 227',
    },
    {
        Id: 228,
        Subject: 'Rank 228',
    },
    {
        Id: 229,
        Subject: 'Rank 229',
    },
    {
        Id: 230,
        Subject: 'Rank 230',
    },
    {
        Id: 231,
        Subject: 'Rank 231',
    },
    {
        Id: 232,
        Subject: 'Rank 232',
    },
    {
        Id: 233,
        Subject: 'Rank 233',
    },
    {
        Id: 234,
        Subject: 'Rank 234',
    },
    {
        Id: 235,
        Subject: 'Rank 235',
    },
    {
        Id: 236,
        Subject: 'Rank 236',
    },
    {
        Id: 237,
        Subject: 'Rank 237',
    },
    {
        Id: 238,
        Subject: 'Rank 238',
    },
    {
        Id: 239,
        Subject: 'Rank 239',
    },
    {
        Id: 240,
        Subject: 'Rank 240',
    },
    {
        Id: 241,
        Subject: 'Rank 241',
    },
    {
        Id: 242,
        Subject: 'Rank 242',
    },
    {
        Id: 243,
        Subject: 'Rank 243',
    },
    {
        Id: 244,
        Subject: 'Rank 244',
    },
    {
        Id: 245,
        Subject: 'Rank 245',
    },
    {
        Id: 246,
        Subject: 'Rank 246',
    },
    {
        Id: 247,
        Subject: 'Rank 247',
    },
    {
        Id: 248,
        Subject: 'Rank 248',
    },
    {
        Id: 249,
        Subject: 'Rank 249',
    },
    {
        Id: 250,
        Subject: 'Rank 250',
    },
    {
        Id: 251,
        Subject: 'Rank 251',
    },
    {
        Id: 252,
        Subject: 'Rank 252',
    },
    {
        Id: 253,
        Subject: 'Rank 253',
    },
    {
        Id: 254,
        Subject: 'Rank 254',
    },
    {
        Id: 255,
        Subject: 'Rank 255',
    },
    {
        Id: 256,
        Subject: 'Rank 256',
    },
    {
        Id: 257,
        Subject: 'Rank 257',
    },
    {
        Id: 258,
        Subject: 'Rank 258',
    },
    {
        Id: 259,
        Subject: 'Rank 259',
    },
    {
        Id: 260,
        Subject: 'Rank 260',
    },
    {
        Id: 261,
        Subject: 'Rank 261',
    },
    {
        Id: 262,
        Subject: 'Rank 262',
    },
    {
        Id: 263,
        Subject: 'Rank 263',
    },
    {
        Id: 264,
        Subject: 'Rank 264',
    },
    {
        Id: 265,
        Subject: 'Rank 265',
    },
    {
        Id: 266,
        Subject: 'Rank 266',
    },
    {
        Id: 267,
        Subject: 'Rank 267',
    },
    {
        Id: 268,
        Subject: 'Rank 268',
    },
    {
        Id: 269,
        Subject: 'Rank 269',
    },
    {
        Id: 270,
        Subject: 'Rank 270',
    },
    {
        Id: 271,
        Subject: 'Rank 271',
    },
    {
        Id: 272,
        Subject: 'Rank 272',
    },
    {
        Id: 273,
        Subject: 'Rank 273',
    },
    {
        Id: 274,
        Subject: 'Rank 274',
    },
    {
        Id: 275,
        Subject: 'Rank 275',
    },
    {
        Id: 276,
        Subject: 'Rank 276',
    },
    {
        Id: 277,
        Subject: 'Rank 277',
    },
    {
        Id: 278,
        Subject: 'Rank 278',
    },
    {
        Id: 279,
        Subject: 'Rank 279',
    },
    {
        Id: 280,
        Subject: 'Rank 280',
    },
    {
        Id: 281,
        Subject: 'Rank 281',
    },
    {
        Id: 282,
        Subject: 'Rank 282',
    },
    {
        Id: 283,
        Subject: 'Rank 283',
    },
    {
        Id: 284,
        Subject: 'Rank 284',
    },
    {
        Id: 285,
        Subject: 'Rank 285',
    },
    {
        Id: 286,
        Subject: 'Rank 286',
    },
    {
        Id: 287,
        Subject: 'Rank 287',
    },
    {
        Id: 288,
        Subject: 'Rank 288',
    },
    {
        Id: 289,
        Subject: 'Rank 289',
    },
    {
        Id: 290,
        Subject: 'Rank 290',
    },
    {
        Id: 291,
        Subject: 'Rank 291',
    },
    {
        Id: 292,
        Subject: 'Rank 292',
    },
    {
        Id: 293,
        Subject: 'Rank 293',
    },
    {
        Id: 294,
        Subject: 'Rank 294',
    },
    {
        Id: 295,
        Subject: 'Rank 295',
    },
    {
        Id: 296,
        Subject: 'Rank 296',
    },
    {
        Id: 297,
        Subject: 'Rank 297',
    },
    {
        Id: 298,
        Subject: 'Rank 298',
    },
    {
        Id: 299,
        Subject: 'Rank 299',
    },
    {
        Id: 300,
        Subject: 'Rank 300',
    },
    {
        Id: 301,
        Subject: 'Rank 301',
    },
    {
        Id: 302,
        Subject: 'Rank 302',
    },
    {
        Id: 303,
        Subject: 'Rank 303',
    },
    {
        Id: 304,
        Subject: 'Rank 304',
    },
    {
        Id: 305,
        Subject: 'Rank 305',
    },
    {
        Id: 306,
        Subject: 'Rank 306',
    },
    {
        Id: 307,
        Subject: 'Rank 307',
    },
    {
        Id: 308,
        Subject: 'Rank 308',
    },
    {
        Id: 309,
        Subject: 'Rank 309',
    },
    {
        Id: 310,
        Subject: 'Rank 310',
    },
    {
        Id: 311,
        Subject: 'Rank 311',
    },
    {
        Id: 312,
        Subject: 'Rank 312',
    },
    {
        Id: 313,
        Subject: 'Rank 313',
    },
    {
        Id: 314,
        Subject: 'Rank 314',
    },
    {
        Id: 315,
        Subject: 'Rank 315',
    },
    {
        Id: 316,
        Subject: 'Rank 316',
    },
    {
        Id: 317,
        Subject: 'Rank 317',
    },
    {
        Id: 318,
        Subject: 'Rank 318',
    },
    {
        Id: 319,
        Subject: 'Rank 319',
    },
    {
        Id: 320,
        Subject: 'Rank 320',
    },
    {
        Id: 321,
        Subject: 'Rank 321',
    },
    {
        Id: 322,
        Subject: 'Rank 322',
    },
    {
        Id: 323,
        Subject: 'Rank 323',
    },
    {
        Id: 324,
        Subject: 'Rank 324',
    },
    {
        Id: 325,
        Subject: 'Rank 325',
    },
    {
        Id: 326,
        Subject: 'Rank 326',
    },
    {
        Id: 327,
        Subject: 'Rank 327',
    },
    {
        Id: 328,
        Subject: 'Rank 328',
    },
    {
        Id: 329,
        Subject: 'Rank 329',
    },
    {
        Id: 330,
        Subject: 'Rank 330',
    },
    {
        Id: 331,
        Subject: 'Rank 331',
    },
    {
        Id: 332,
        Subject: 'Rank 332',
    },
    {
        Id: 333,
        Subject: 'Rank 333',
    },
    {
        Id: 334,
        Subject: 'Rank 334',
    },
    {
        Id: 335,
        Subject: 'Rank 335',
    },
    {
        Id: 336,
        Subject: 'Rank 336',
    },
    {
        Id: 337,
        Subject: 'Rank 337',
    },
    {
        Id: 338,
        Subject: 'Rank 338',
    },
    {
        Id: 339,
        Subject: 'Rank 339',
    },
    {
        Id: 340,
        Subject: 'Rank 340',
    },
    {
        Id: 341,
        Subject: 'Rank 341',
    },
    {
        Id: 342,
        Subject: 'Rank 342',
    },
    {
        Id: 343,
        Subject: 'Rank 343',
    },
    {
        Id: 344,
        Subject: 'Rank 344',
    },
    {
        Id: 345,
        Subject: 'Rank 345',
    },
    {
        Id: 346,
        Subject: 'Rank 346',
    },
    {
        Id: 347,
        Subject: 'Rank 347',
    },
    {
        Id: 348,
        Subject: 'Rank 348',
    },
    {
        Id: 349,
        Subject: 'Rank 349',
    },
    {
        Id: 350,
        Subject: 'Rank 350',
    },
    {
        Id: 351,
        Subject: 'Rank 351',
    },
    {
        Id: 352,
        Subject: 'Rank 352',
    },
    {
        Id: 353,
        Subject: 'Rank 353',
    },
    {
        Id: 354,
        Subject: 'Rank 354',
    },
    {
        Id: 355,
        Subject: 'Rank 355',
    },
    {
        Id: 356,
        Subject: 'Rank 356',
    },
    {
        Id: 357,
        Subject: 'Rank 357',
    },
    {
        Id: 358,
        Subject: 'Rank 358',
    },
    {
        Id: 359,
        Subject: 'Rank 359',
    },
    {
        Id: 360,
        Subject: 'Rank 360',
    },
    {
        Id: 361,
        Subject: 'Rank 361',
    },
    {
        Id: 362,
        Subject: 'Rank 362',
    },
    {
        Id: 363,
        Subject: 'Rank 363',
    },
    {
        Id: 364,
        Subject: 'Rank 364',
    },
    {
        Id: 365,
        Subject: 'Rank 365',
    },
    {
        Id: 366,
        Subject: 'Rank 366',
    },
    {
        Id: 367,
        Subject: 'Rank 367',
    },
    {
        Id: 368,
        Subject: 'Rank 368',
    },
    {
        Id: 369,
        Subject: 'Rank 369',
    },
    {
        Id: 370,
        Subject: 'Rank 370',
    },
    {
        Id: 371,
        Subject: 'Rank 371',
    },
    {
        Id: 372,
        Subject: 'Rank 372',
    },
    {
        Id: 373,
        Subject: 'Rank 373',
    },
    {
        Id: 374,
        Subject: 'Rank 374',
    },
    {
        Id: 375,
        Subject: 'Rank 375',
    },
    {
        Id: 376,
        Subject: 'Rank 376',
    },
    {
        Id: 377,
        Subject: 'Rank 377',
    },
    {
        Id: 378,
        Subject: 'Rank 378',
    },
    {
        Id: 379,
        Subject: 'Rank 379',
    },
    {
        Id: 380,
        Subject: 'Rank 380',
    },
    {
        Id: 381,
        Subject: 'Rank 381',
    },
    {
        Id: 382,
        Subject: 'Rank 382',
    },
    {
        Id: 383,
        Subject: 'Rank 383',
    },
    {
        Id: 384,
        Subject: 'Rank 384',
    },
    {
        Id: 385,
        Subject: 'Rank 385',
    },
    {
        Id: 386,
        Subject: 'Rank 386',
    },
    {
        Id: 387,
        Subject: 'Rank 387',
    },
    {
        Id: 388,
        Subject: 'Rank 388',
    },
    {
        Id: 389,
        Subject: 'Rank 389',
    },
    {
        Id: 390,
        Subject: 'Rank 390',
    },
    {
        Id: 391,
        Subject: 'Rank 391',
    },
    {
        Id: 392,
        Subject: 'Rank 392',
    },
    {
        Id: 393,
        Subject: 'Rank 393',
    },
    {
        Id: 394,
        Subject: 'Rank 394',
    },
    {
        Id: 395,
        Subject: 'Rank 395',
    },
    {
        Id: 396,
        Subject: 'Rank 396',
    },
    {
        Id: 397,
        Subject: 'Rank 397',
    },
    {
        Id: 398,
        Subject: 'Rank 398',
    },
    {
        Id: 399,
        Subject: 'Rank 399',
    },
    {
        Id: 400,
        Subject: 'Rank 400',
    },
    {
        Id: 401,
        Subject: 'Rank 401',
    },
    {
        Id: 402,
        Subject: 'Rank 402',
    },
    {
        Id: 403,
        Subject: 'Rank 403',
    },
    {
        Id: 404,
        Subject: 'Rank 404',
    },
    {
        Id: 405,
        Subject: 'Rank 405',
    },
    {
        Id: 406,
        Subject: 'Rank 406',
    },
    {
        Id: 407,
        Subject: 'Rank 407',
    },
    {
        Id: 408,
        Subject: 'Rank 408',
    },
    {
        Id: 409,
        Subject: 'Rank 409',
    },
    {
        Id: 410,
        Subject: 'Rank 410',
    },
    {
        Id: 411,
        Subject: 'Rank 411',
    },
    {
        Id: 412,
        Subject: 'Rank 412',
    },
    {
        Id: 413,
        Subject: 'Rank 413',
    },
    {
        Id: 414,
        Subject: 'Rank 414',
    },
    {
        Id: 415,
        Subject: 'Rank 415',
    },
    {
        Id: 416,
        Subject: 'Rank 416',
    },
    {
        Id: 417,
        Subject: 'Rank 417',
    },
    {
        Id: 418,
        Subject: 'Rank 418',
    },
    {
        Id: 419,
        Subject: 'Rank 419',
    },
    {
        Id: 420,
        Subject: 'Rank 420',
    },
    {
        Id: 421,
        Subject: 'Rank 421',
    },
    {
        Id: 422,
        Subject: 'Rank 422',
    },
    {
        Id: 423,
        Subject: 'Rank 423',
    },
    {
        Id: 424,
        Subject: 'Rank 424',
    },
    {
        Id: 425,
        Subject: 'Rank 425',
    },
    {
        Id: 426,
        Subject: 'Rank 426',
    },
    {
        Id: 427,
        Subject: 'Rank 427',
    },
    {
        Id: 428,
        Subject: 'Rank 428',
    },
    {
        Id: 429,
        Subject: 'Rank 429',
    },
    {
        Id: 430,
        Subject: 'Rank 430',
    },
    {
        Id: 431,
        Subject: 'Rank 431',
    },
    {
        Id: 432,
        Subject: 'Rank 432',
    },
    {
        Id: 433,
        Subject: 'Rank 433',
    },
    {
        Id: 434,
        Subject: 'Rank 434',
    },
    {
        Id: 435,
        Subject: 'Rank 435',
    },
    {
        Id: 436,
        Subject: 'Rank 436',
    },
    {
        Id: 437,
        Subject: 'Rank 437',
    },
    {
        Id: 438,
        Subject: 'Rank 438',
    },
    {
        Id: 439,
        Subject: 'Rank 439',
    },
    {
        Id: 440,
        Subject: 'Rank 440',
    },
    {
        Id: 441,
        Subject: 'Rank 441',
    },
    {
        Id: 442,
        Subject: 'Rank 442',
    },
    {
        Id: 443,
        Subject: 'Rank 443',
    },
    {
        Id: 444,
        Subject: 'Rank 444',
    },
    {
        Id: 445,
        Subject: 'Rank 445',
    },
    {
        Id: 446,
        Subject: 'Rank 446',
    },
    {
        Id: 447,
        Subject: 'Rank 447',
    },
    {
        Id: 448,
        Subject: 'Rank 448',
    },
    {
        Id: 449,
        Subject: 'Rank 449',
    },
    {
        Id: 450,
        Subject: 'Rank 450',
    },
    {
        Id: 451,
        Subject: 'Rank 451',
    },
    {
        Id: 452,
        Subject: 'Rank 452',
    },
    {
        Id: 453,
        Subject: 'Rank 453',
    },
    {
        Id: 454,
        Subject: 'Rank 454',
    },
    {
        Id: 455,
        Subject: 'Rank 455',
    },
    {
        Id: 45
```



```

        StartTime: new Date(2017, 9, 29, 10, 0),
        EndTime: new Date(2017, 9, 29, 11, 30),
        IsAllDay: false,
        RankId: '1'
    }, {
        Id: 2,
        Subject: 'Rank 3',
        StartTime: new Date(2017, 9, 29, 10, 30),
        EndTime: new Date(2017, 9, 29, 12, 30),
        IsAllDay: false,
        RankId: '3'
    }, {
        Id: 3,
        Subject: 'Rank 6',
        StartTime: new Date(2017, 9, 29, 7, 0),
        EndTime: new Date(2017, 9, 29, 14, 30),
        IsAllDay: false,
        RankId: '6'
    }, {
        Id: 4,
        Subject: 'Rank 9',
        StartTime: new Date(2017, 9, 29, 11, 0),
        EndTime: new Date(2017, 9, 29, 15, 30),
        IsAllDay: false,
        RankId: '9'
    }
    ];
    const eventSettings = { dataSource: data, sortComparer: comparerFun }
    return <ScheduleComponent height='550px' selectedDate={new Date(2017, 9,
29)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
</>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
    const comparerFun = (args: Record<string, any>[]) => {
        args.sort((event1: Record<string, any>, event2: Record<string, any>) =>
event1.RankId.localeCompare(event2.RankId, undefined, { numeric: true }));
        return args;
    }
    const data: Object[] = [{
        Id: 1,
        Subject: 'Rank 1',
        StartTime: new Date(2017, 9, 29, 10, 0),
        EndTime: new Date(2017, 9, 29, 11, 30),
        IsAllDay: false,
        RankId: '1'
    }
    ];

```

```

    }, {
      Id: 2,
      Subject: 'Rank 3',
      StartTime: new Date(2017, 9, 29, 10, 30),
      EndTime: new Date(2017, 9, 29, 12, 30),
      IsAllDay: false,
      RankId: '3'
    }, {
      Id: 3,
      Subject: 'Rank 6',
      StartTime: new Date(2017, 9, 29, 7, 0),
      EndTime: new Date(2017, 9, 29, 14, 30),
      IsAllDay: false,
      RankId: '6'
    }, {
      Id: 4,
      Subject: 'Rank 9',
      StartTime: new Date(2017, 9, 29, 11, 0),
      EndTime: new Date(2017, 9, 29, 15, 30),
      IsAllDay: false,
      RankId: '9'
    }
  ];
  const eventSettings: EventSettingsModel = { dataSource: data, sortComparer:
  comparerFun };
  return <ScheduleComponent height='550px' selectedDate={new Date(2017, 9,
  29)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
  };
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
  base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
  buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
  calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
  dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
  inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
  navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
  popups/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Drag and drop appointments

Appointments can be rescheduled to any time by dragging and dropping them onto the desired location. To work with drag and drop functionality, it is necessary to inject the module **DragAndDrop** and make sure that [Link to the Video](#) is set to true on Scheduler. In mobile mode, you can drag and drop the events by tap holding an event and dropping them on to the desired location.

Learn the advanced options of Drag and Resize actions for React Scheduler from this video:

By default, drag and drop action is applicable on all Scheduler views, except Agenda, Month-Agenda and Year view.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, TimelineViews, TimelineMonth, Month,
Agenda, DragAndDrop, ViewsDirective, ViewDirective, Inject } from
'@syncfusion/ej2-react-schedule';

```

```
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineMonth' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews, TimelineMonth, Month, Agenda, DragAndDrop]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, TimelineViews, TimelineMonth, Month, Agenda,
  DragAndDrop, ViewsDirective, ViewDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineMonth' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews, TimelineMonth, Month, Agenda, DragAndDrop]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta name="description" content="Essential JS 2 for React Components" />
<meta name="author" content="Syncfusion" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Drag and drop multiple appointments

We can drag and drop multiple appointments by enabling the `allowMultiDrag` property. We can select multiple appointments by holding the CTRL key. Once the events are selected, we can leave the CTRL key and start dragging the event.

We can also drag multiple events from one resource to another resource. In this case, if all the selected events are in the different resources, then all the events should be moved to the single resource that is related to the target event.

Note: Multiple events drag and drop is not supported on mobile devices.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} allowMultiDrag={true} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData }
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} allowMultiDrag={true} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
    </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

```

    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <link href="index.css" rel="stylesheet" />
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
        .e-past-app {
            background-color: chocolate !important;
        }
        .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
        .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Disable the drag action

By default, you can drag and drop the events within any of the applicable scheduler views, and to disable it, set `false` to the `allowDragAndDrop` property.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} allowDragAndDrop={false} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData }
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} allowDragAndDrop={false} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
    </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
```



```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Preventing drag and drop on specific targets

It is possible to prevent the drag action on particular target, by passing the target to be excluded in the `excludeSelectors` option within `dragStart` event. In the following example, we have prevented the drag action on all-day row.

INDEX.JSX

```
import * as React from 'react';
```

```
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onDragStart = (args) => {
    args.excludeSelectors = 'e-header-cells,e-header-day,e-header-date,e-all-
day-cells';
  }
  const eventSettings = { dataSource: scheduleData };
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} dragStart={onDragStart}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Inject,
  DragEventArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onDragStart = (args: DragEventArgs): void => {
    args.excludeSelectors = 'e-header-cells,e-header-day,e-header-date,e-all-
day-cells';
  }
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} dragStart={onDragStart}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Disable scrolling on drag action

By default, while dragging an appointment to the edges, either top or bottom of the Scheduler, scrolling action takes place automatically. To prevent this scrolling, set `false` to the `scroll` value within the `dragStart` event arguments.

INDEX.JSX

```
import * as React from 'react';
```

```
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onDragStart = (args) => {
    args.scroll = { enable: false };
  }
  const eventSettings = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} dragStart={onDragStart}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Inject,
  DragEventArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onDragStart = (args: DragEventArgs): void => {
    args.scroll = { enable: false };
  }
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} dragStart={onDragStart}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008c9f;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Controlling scroll speed while dragging an event

The speed of the scrolling action while dragging an appointment to the Scheduler edges, can be controlled within the `dragStart` event by setting the desired value to the `scrollBy` and `timeDelay` option whereas its default value is 30 minutes and 100ms.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onDragStart = (args) => {
    args.scroll = { enable: true, scrollBy: 5, timeDelay: 200 };
  }
  const eventSettings = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} dragStart={onDragStart}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Inject,
  DragEventArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onDragStart = (args: DragEventArgs): void => {
    args.scroll = { enable: true, scrollBy: 5, timeDelay: 200 };
  }
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} dragStart={onDragStart}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008c8f;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Auto navigation of date ranges on dragging an event

When an event is dragged either to the left or right extreme edges of the Scheduler and kept hold for few seconds without dropping, the auto navigation of date ranges will be enabled allowing the Scheduler to navigate from current date range to back and forth respectively. This action is set to `false` by default and to enable it, you need to set `navigation` to true within the `dragStart` event.

By default, the navigation delay is set to 2000ms. The navigation delay decides how long the user needs to drag and hold the appointments at the extremities. You can also set your own delay value for letting the users to navigate based on it, using the `timeDelay` within the `dragStart` event.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const onDragStart = (args) => {
        args.navigation = { enable: true, timeDelay: 4000 };
    }
    const eventSettings = { dataSource: scheduleData }
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} dragStart={onDragStart}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Inject,
    DragEventArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const onDragStart = (args: DragEventArgs): void => {
        args.navigation = { enable: true, timeDelay: 4000 }
    }
    const eventSettings: EventSettingsModel = { dataSource: scheduleData }
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} dragStart={onDragStart}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
    </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
```



```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Setting drag time interval

By default, while dragging an appointment, it moves at an interval of 30 minutes. To change the dragging time interval, pass the appropriate values to the `interval` option within the `dragStart` event.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onDragStart = (args) => {
    args.interval = 10;
  }
  const eventSettings = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} dragStart={({ onDragStart })}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Inject,
  DragEventArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onDragStart = (args: DragEventArgs): void => {
    args.interval = 10;
  }
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} dragStart={({ onDragStart })}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Drag and drop items from external source

It is possible to drag and drop the unplanned items from any of the external source into the scheduler, by manually saving those dropped item as a new appointment data through `addEvent` method of Scheduler.

In this example, we have used the tree view control as an external source and the child nodes from the tree view component are dragged and dropped onto the Scheduler. Therefore, it is necessary to make

use of the `nodeDragStop` event of the `TreeView` component, where we can form an event object and save it using the [Link to the Video](#) method.

Learn how to drag an external item into the React Scheduler by watching this video:

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Inject,
TimelineViews, Resize, DragAndDrop } from '@syncfusion/ej2-react-schedule';
import { eventData, waitingList } from './datasource';
import { closest, remove, addClass } from '@syncfusion/ej2-base';
import { TreeViewComponent } from '@syncfusion/ej2-react-navigations';
const App = () => {
    const scheduleObj = useRef(null);
    const treeObj = useRef(null);
    let isTreeItemDropped = false;
    let draggedItemId = '';
    let allowDragAndDrops = true;
    const fields = { dataSource: waitingList, id: 'Id', text: 'Name' };
    const fieldsData = {
        subject: { title: 'Patient Name', name: 'Name' },
        startTime: { title: "From", name: "StartTime" },
        endTime: { title: "To", name: "EndTime" },
        description: { title: 'Reason', name: 'Description' }
    }
    const eventSettings = { dataSource: eventData, fields: fieldsData };
    const treeTemplate = (props) => {
        return (<div id="waiting"><div id="waitdetails"><div
id="waitlist">{props.Name}</div>
        <div id="waitcategory">{props.DepartmentName}</div></div></div>);
    }
    const onItemDrag = (event) => {
        if (scheduleObj.current.isAdaptive) {
            let classElement = scheduleObj.current.element.querySelector('.e-device-hover');
            if (classElement) {
                classElement.classList.remove('e-device-hover');
            }
            if (event.target.classList.contains('e-work-cells')) {
                addClass([event.target], 'e-device-hover');
            }
        }
        if (document.body.style.cursor === 'not-allowed') {
            document.body.style.cursor = '';
        }
        if (event.name === 'nodeDragging') {
            let dragElementIcon = document.querySelectorAll('.e-drag-item.treeview-external-drag .e-icon-expandable');
            for (let i = 0; i < dragElementIcon.length; i++) {
                dragElementIcon[i].style.display = 'none';
            }
        }
    }
    const onActionBegin = (event) => {
```

```

    if (event.requestType === 'eventCreate' && isTreeItemDropped) {
      let treeViewdata = treeObj.current.fields.dataSource;
      const filteredPeople = treeViewdata.filter((item) => item.Id !==
parseInt(draggedItemId, 10));
      treeObj.current.fields.dataSource = filteredPeople;
      let elements = document.querySelectorAll('.e-drag-item.treeview-
external-drag');
      for (let i = 0; i < elements.length; i++) {
        remove(elements[i]);
      }
    }
  }
  const onTreeDragStop = (event) => {
    let treeElement = closest(event.target, '.e-treeview');
    let classElement = scheduleObj.current.element.querySelector('.e-device-
hover');
    if (classElement) {
      classElement.classList.remove('e-device-hover');
    }
    if (!treeElement) {
      event.cancel = true;
      let scheduleElement = closest(event.target, '.e-content-wrap');
      if (scheduleElement) {
        let treeviewData = treeObj.current.fields.dataSource;
        if (event.target.classList.contains('e-work-cells')) {
          const filteredData = treeviewData.filter((item) => item.Id ===
parseInt(event.draggedNodeData.id, 10));
          let cellData = scheduleObj.current.getCellDetails(event.target);
          let eventData = {
            Name: filteredData[0].Name,
            StartTime: cellData.startTime,
            EndTime: cellData.endTime,
            IsAllDay: cellData.isAllDay,
            Description: filteredData[0].Description
          };
          scheduleObj.current.addEvent(eventData);
          isTreeItemDropped = true;
          draggedItemId = event.draggedNodeData.id;
        }
      }
    }
  }
}
return (<div className='schedule-control-section'>
  <div className='col-lg-12 control-section'>
    <div className='content-wrapper'>
      <div className="schedule-container">
        <div className="title-container">
          <div className="title-text">Scheduler</div>
        </div>
        <ScheduleComponent ref={scheduleObj} cssClass='schedule-drag-drop'
width='100%' height='650px' selectedDate={new Date(2018, 7, 1)}
currentView='TimelineDay' eventSettings={eventSettings}
actionBegin={onActionBegin} drag={onItemDrag}>
          <ViewsDirective>
            <ViewDirective option='TimelineDay' />
          </ViewsDirective>
          <Inject services={[TimelineViews, Resize, DragAndDrop]} />
        </div>
      </div>
    </div>
  </div>
</div>

```

```

        </ScheduleComponent>
      </div>
      <div className="treeview-container">
        <div className="title-container">
          <div className="title-text">Waiting List</div>
        </div>
        <TreeViewComponent ref={treeObj} cssClass='treeview-external-drag'
nodeTemplate={treeTemplate} fields={fields} nodeDragStop={onTreeDragStop}
nodeDragging={onItemDrag} allowDragAndDrop={allowDragAndDrops} />
      </div>
    </div>
  </div>
</div>);
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import {
  ScheduleComponent, ViewsDirective, ViewDirective, Inject, TimelineViews,
  Resize, DragAndDrop, ActionEventArgs, CellClickEventArgs,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { eventData, waitingList } from '../datasource';
import { closest, remove, addClass } from '@syncfusion/ej2-base';
import { DragAndDropEventArgs } from '@syncfusion/ej2-navigations';
import { TreeViewComponent } from '@syncfusion/ej2-react-navigations';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const treeObj = useRef<TreeViewComponent>(null);
  let isTreeItemDropped: boolean = false;
  let draggedItemId: string = '';
  let allowDragAndDrops: boolean = true;
  const fields: Object = { dataSource: waitingList, id: 'Id', text: 'Name' };
  const fieldsData = {
    subject: { title: 'Patient Name', name: 'Name' },
    startTime: { title: "From", name: "StartTime" },
    endTime: { title: "To", name: "EndTime" },
    description: { title: 'Reason', name: 'Description' }
  }
  const eventSettings: EventSettingsModel = { dataSource: eventData, fields:
fieldsData };
  const treeTemplate = (props: any): JSX.Element => {
    return (<div id="waiting"><div id="waitdetails"><div
id="waitlist">{props.Name}</div>
      <div id="waitcategory">{props.DepartmentName}</div></div></div>);
  }
  const onItemDrag = (event: any): void => {
    if (scheduleObj.current.isAdaptive) {
      let classElement: HTMLElement =
scheduleObj.current.element.querySelector('.e-device-hover');
      if (classElement) {

```

```

        classElement.classList.remove('e-device-hover');
    }
    if (event.target.classList.contains('e-work-cells')) {
        addClass([event.target], 'e-device-hover');
    }
}
if (document.body.style.cursor === 'not-allowed') {
    document.body.style.cursor = '';
}
if (event.name === 'nodeDragging') {
    let dragElementIcon: NodeListOf<HTMLElement> =
        document.querySelectorAll('.e-drag-item.treeview-external-drag .e-
icon-expandable');
    for (let i: number = 0; i < dragElementIcon.length; i++) {
        dragElementIcon[i].style.display = 'none';
    }
}
}
const onActionBegin = (event: ActionEventArgs): void => {
    if (event.requestType === 'eventCreate' && isTreeItemDropped) {
        let treeViewdata: { [key: string]: Object }[] =
treeObj.current.fields.dataSource as { [key: string]: Object }[];
        const filteredPeople: { [key: string]: Object }[] =
            treeViewdata.filter((item: any) => item.Id !==
parseInt(draggedItemId, 10));
        treeObj.current.fields.dataSource = filteredPeople;
        let elements: NodeListOf<HTMLElement> = document.querySelectorAll('.e-
drag-item.treeview-external-drag');
        for (let i: number = 0; i < elements.length; i++) {
            remove(elements[i]);
        }
    }
}
const onTreeDragStop = (event: DragAndDropEventArgs): void => {
    let treeElement: Element = closest(event.target, '.e-treeview');
    let classElement: HTMLElement =
scheduleObj.current.element.querySelector('.e-device-hover');
    if (classElement) {
        classElement.classList.remove('e-device-hover');
    }
    if (!treeElement) {
        event.cancel = true;
        let scheduleElement: Element = closest(event.target, '.e-content-
wrap');
        if (scheduleElement) {
            let treeviewData: { [key: string]: Object }[] =
                treeObj.current.fields.dataSource as { [key: string]: Object }[];
            if (event.target.classList.contains('e-work-cells')) {
                const filteredData: { [key: string]: Object }[] =
                    treeviewData.filter((item: any) => item.Id ===
parseInt(event.draggedNodeData.id as string, 10));
                let cellData: CellClickEventArgs =
scheduleObj.current.getCellDetails(event.target);
                let eventData: { [key: string]: Object } = {
                    Name: filteredData[0].Name,
                    StartTime: cellData.startTime,
                    EndTime: cellData.endTime,

```

```

        IsAllDay: cellData.isAllDay,
        Description: filteredData[0].Description
    };
    scheduleObj.current.addEvent(eventData);
    isTreeItemDropped = true;
    draggedItemId = event.draggedNodeData.id as string;
    }
    }
    }
    }
    return (
        <div className='schedule-control-section'>
            <div className='col-lg-12 control-section'>
                <div className='content-wrapper'>
                    <div className="schedule-container">
                        <div className="title-container">
                            <div className="title-text">Scheduler</div>
                        </div>
                        <ScheduleComponent ref={scheduleObj} cssClass='schedule-drag-
drop' width='100%' height='650px' selectedDate={new Date(2018, 7, 1)}
                            currentView='TimelineDay'
                            eventSettings={eventSettings}
                            actionBegin={onActionBegin} drag={onItemDrag} >
                                <ViewsDirective>
                                    <ViewDirective option='TimelineDay' />
                                </ViewsDirective>
                                <Inject services={[TimelineViews, Resize, DragAndDrop]} />
                            </ScheduleComponent>
                        </div>
                        <div className="treeview-container">
                            <div className="title-container">
                                <div className="title-text">Waiting List</div>
                            </div>
                            <TreeViewComponent ref={treeObj} cssClass='treeview-external-
drag' nodeTemplate={treeTemplate} fields={fields}
                                nodeDragStop={onTreeDragStop} nodeDragging={onItemDrag}
                                allowDragAndDrop={allowDragAndDrops} />
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        );
    }
    const root = ReactDOM.createRoot(document.getElementById('schedule'));
    root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />

```



```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008c8f;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Opening the editor window on drag stop

There are scenarios where you want to open the editor filled with data on newly dropped location and may need to proceed to save it, only when **Save** button is clicked on the editor. On clicking the cancel button should revert these changes. This can be achieved using the **dragStop** event of Scheduler.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {

```

```

const scheduleObj = useRef(null);
const onDragStop = (args) => {
  args.cancel = true; //cancels the drop action
  scheduleObj.current.openEditor(args.data, "Save"); //open the event window with updated start and end time
}
const eventSettings = { dataSource: scheduleData }
return <ScheduleComponent height='550px' ref={scheduleObj}
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
dragStop={onDragStop}>
  <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
</ScheduleComponent>;
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Inject,
  DragEventArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const onDragStop = (args: DragEventArgs): void => {
    args.cancel = true; //cancels the drop action
    scheduleObj.current.openEditor(args.data, "Save"); //open the event window with updated start and end time
  }
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' ref={scheduleObj}
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
dragStop={onDragStop}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Inline Appointment

In Scheduler, another easier way for **adding** or **editing** the appointment's subject alone can be achieved by using the inline Add/Edit support. It allows the user to add and edit the appointments inline. To get familiar with the inline Add mode, single click on any of the Scheduler cells or press enter key on the selected cells.

When the inline adding mode is ON, a text box will get created within the clicked Scheduler cells with a blinking cursor in it, requiring the user to enter the subject of an appointment. Once the subject is entered, the appointment will be saved on pressing the enter key.

To enable the inline edit mode, single click on any of the existing appointment's subject, so that the user can edit the subject of that appointment. The edited subject of that appointment will be updated on pressing the enter key.

The inline option can be enabled/disabled on the Scheduler by using the `allowInline` API, whereas its default value is set to `false`.

While using the `allowInline` the `showQuickInfo` will be turned off. The `quickPopup` will not show on clicking the work cell or clicking the appointment when the `allowInline` property is set to `true`.

In work cells, select multiple cells using keyboard, and then press enter key. The appointment wrapper will be created, and focus will be on the subject field. Also, consider the overlapping scenarios when creating an inline event.

Normal Event

While editing appointments, single-click the appointment subject, the `editable` option will be enabled in UI and the cursor will focus at the end of the text. Inline editing will be considered for all possible views.

Recurrence Event

While editing the occurrence from the recurrence series, it is only possible to edit a `single occurrence`, not an entire series.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Agenda, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData }
  return <ScheduleComponent width='100%' height='550px' currentView='Month'
selectedDate={new Date(2018, 1, 17)} allowInline={true}
eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Agenda, Month]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
```

```
import {
  ScheduleComponent, Day, Week, WorkWeek, Agenda, Month, Inject,
  EventSettingsModel,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent width='100%' height='550px' currentView='Month'
    selectedDate={new Date(2018, 1, 17)} allowInline={true}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Agenda, Month]} />
  </ScheduleComponent>
</>;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
```

```

<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Appointment Resizing

Another way of rescheduling an appointment can be done by resizing it through either of its handlers. To work with resizing functionality, it is necessary to inject the module `Resize` and make sure that `allowResizing` property is set to true.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, TimelineViews, TimelineMonth, Month,
Agenda, Resize, ViewsDirective, ViewDirective, Inject } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='TimelineWeek' />
            <ViewDirective option='Week' />
            <ViewDirective option='TimelineMonth' />
            <ViewDirective option='Month' />
            <ViewDirective option='Agenda' />
        </ViewsDirective>
    </ScheduleComponent>

```

```

    <Inject services={[Day, Week, TimelineViews, TimelineMonth, Month,
    Agenda, Resize]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, TimelineViews, TimelineMonth, Month, Agenda,
  Resize, ViewsDirective, ViewDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineMonth' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews, TimelineMonth, Month,
    Agenda, Resize]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Disable the resize action

By default, resizing of events is allowed on all Scheduler views except Agenda and Month-Agenda view. To disable this event resizing action, set false to the `allowResizing` property.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Resize,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }

```



```

    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} allowResizing={false} eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda, Resize]} />
    </ScheduleComponent>;
  }
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Resize, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} allowResizing={false} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, Resize]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />

```

```

    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <link href="index.css" rel="stylesheet" />
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008c9f;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
        .e-past-app {
            background-color: chocolate !important;
        }
        .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
        .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Disable scrolling on resize action

By default, while resizing an appointment, when its handler reaches the extreme edges of the Scheduler, scrolling action will take place along with event resizing. To prevent this scrolling action, set false to **scroll** value within the **resizeStart** event.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Resize,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const onResizeStart = (args) => {
        args.scroll = { enable: false };
    }
    const eventSettings = { dataSource: scheduleData }
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} resizeStart={onResizeStart}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda, Resize]} />
    </ScheduleComponent>

```

```

    </ScheduleComponent>;
  }
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Resize,
  ResizeEventArgs, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onResizeStart = (args: ResizeEventArgs): void => {
    args.scroll = { enable: false };
  }
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} resizeStart={onResizeStart}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, Resize]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />

```

```

    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <link href="index.css" rel="stylesheet" />
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
        .e-past-app {
            background-color: chocolate !important;
        }
        .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
        .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Controlling scroll speed while resizing an event

The speed of the scrolling action while resizing an appointment to the Scheduler edges, can be controlled within the `resizeStart` event by setting the desired value to the `scrollBy` option.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Resize,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const onResizeStart = (args) => {
        args.scroll = { enable: true, scrollBy: 15 };
    }
    const eventSettings = { dataSource: scheduleData }
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} resizeStart={onResizeStart}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda, Resize]} />
    </ScheduleComponent>;
}

```

```

}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Resize,
  ResizeEventArgs, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onResizeStart = (args: ResizeEventArgs): void => {
    args.scroll = { enable: true, scrollBy: 15 };
  }
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} resizeStart={onResizeStart}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, Resize]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />

```

```

<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
      background: green;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Setting resize time interval

By default, while resizing an appointment, it extends or shrinks at an interval of 30 minutes. To change this default resize interval, set appropriate values to `interval` option within the `resizeStart` event.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Resize,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onResizeStart = (args) => {
    args.interval = 10;
  }
  const eventSettings = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} resizeStart={{onResizeStart}}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, Resize]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));

```

```
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Resize,
  ResizeEventArgs, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const onResizeStart = (args: ResizeEventArgs): void => {
    args.interval = 10;
  }
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings} resizeStart={onResizeStart}>>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, Resize]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
```

```

<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Appointment customization

The look and feel of the Scheduler events can be customized using any one of the following ways.

- [Using event templates](#)
- [Using eventRendered event](#)
- [Using customCSS class](#)

Using template

Any kind of text, images and links can be added to customize the look of the events. The user can format and change the default appearance of the events by making use of the `template` option available within the [Link to the Video](#) property. The following code example customizes the appointment's default color and time format.

Learn how easily you can customize the basic look and feel of React Scheduler appointments using template from this video:

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject
} from '@syncfusion/ej2-react-schedule';
import { webinarData } from './datasource';

```



```
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const instance = new Internationalization();
  const getTimeString = (value) => {
    return instance.formatDate(value, { skeleton: 'hm' });
  }
  const eventTemplate = (props) => {
    const secondaryColor = { background: props.SecondaryColor };
    const primaryColor_1 = { background: props.PrimaryColor };
    const primaryColor_2 = { background: props.PrimaryColor };
    return (<div className="template-wrap" style={secondaryColor}>
      <div className="subject" style={primaryColor_1}>{props.Subject}</div>
      <div className="time" style={primaryColor_2}>
        Time: {getTimeString(props.StartTime)} -
      {getTimeString(props.EndTime)}</div>
    </div>);
  }
  const eventSettings = { dataSource: webinarData, template: eventTemplate };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} readonly={true}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { webinarData } from '../datasource';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const instance: Internationalization = new Internationalization();
  const getTimeString = (value: Date) => {
    return instance.formatDate(value, { skeleton: 'hm' });
  }
  const eventTemplate = (props) => {
    const secondaryColor = { background: props.SecondaryColor };
    const primaryColor_1 = { background: props.PrimaryColor };
    const primaryColor_2 = { background: props.PrimaryColor };
    return (<div className="template-wrap" style={secondaryColor}>
      <div className="subject" style={primaryColor_1}>{props.Subject}</div>
      <div className="time" style={primaryColor_2}>
        Time: {getTimeString(props.StartTime)} -
      {getTimeString(props.EndTime)}</div>
    </div>);
  }
  const eventSettings: EventSettingsModel = { dataSource: webinarData,
template: eventTemplate };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} readonly={true}>
```

```

    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>

```

```
</html>
```

All the built-in fields that are mapped to the appropriate field properties within the `eventSettings`, as well as custom mapped fields from the Scheduler dataSource can be accessed within the template code.

Using `eventRendered` event

The `eventRendered` event triggers before the appointment renders on the Scheduler. Therefore, this client-side event can be utilized to customize the look of events based on any specific criteria, before rendering them on the scheduler.

INDEX.JSX

```
import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const scheduleObj = useRef(null);
    const eventSettings = { dataSource: scheduleData };
    const onEventRendered = (args) => {
        applyCategoryColor(args, scheduleObj.current.currentView);
    }
    const applyCategoryColor = (args, currentView) => {
        let categoryColor = args.data.CategoryColor;
        if (!args.element || !categoryColor) {
            return;
        }
        if (currentView === 'Agenda') {
            args.element.firstChild.style.borderColor = categoryColor;
        }
        else {
            args.element.style.backgroundColor = categoryColor;
        }
    }
    return (
        <ScheduleComponent width='100%' height='550px' ref={scheduleObj}
        selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
        eventRendered={onEventRendered}>
            <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
    );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    EventRenderedArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
```

```
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  const onEventRendered = (args: EventRenderedArgs): void => {
    applyCategoryColor(args, scheduleObj.current.currentView);
  }
  const applyCategoryColor = (args: EventRenderedArgs, currentView: string):
void => {
    let categoryColor: string = args.data.CategoryColor as string;
    if (!args.element || !categoryColor) {
      return;
    }
    if (currentView === 'Agenda') {
      (args.element.firstChild as HTMLElement).style.borderLeftColor =
categoryColor;
    } else {
      args.element.style.backgroundColor = categoryColor;
    }
  }
  return (
    <ScheduleComponent width='100%' height='550px' ref={scheduleObj}
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
eventRendered={onEventRendered}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Using cssClass

The customization of events can also be achieved using `cssClass` property of the Scheduler. In the following example, the background of appointments has been changed using the `cssClass`.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    return (<ScheduleComponent width='100%' height='550px' cssClass='custom-
class' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}

```

```
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return (
    <ScheduleComponent width='100%' height='550px' cssClass='custom-class'
    selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
```

```

<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Setting minimum height

It is possible to set minimal height for appointments on Scheduler using `eventRendered` event, when its start and end time duration is less than the default duration of a single slot.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject
} from '@syncfusion/ej2-react-schedule';
const App = () => {
    const scheduleObj = useRef(null);
    const data = [{
        Id: 13,
        Subject: 'Myths of Andromeda Galaxy',
        StartTime: new Date(2018, 1, 16, 10, 30),
        EndTime: new Date(2018, 1, 16, 10, 40)
    }, {
        Id: 14,
        Subject: 'Aliens vs Humans',
        StartTime: new Date(2018, 1, 15, 10, 0),

```

```

        EndTime: new Date(2018, 1, 15, 10, 20)
    }];
    const eventSettings = { dataSource: data }
    const onEventRendered = (args) => {
        let cellHeight = scheduleObj.current.element.querySelector('.e-work-
cells').offsetHeight;
        let appHeight = (args.data.EndTime.getTime() -
args.data.StartTime.getTime()) / (60 * 1000) * (cellHeight *
scheduleObj.current.timeScale.slotCount) /
scheduleObj.current.timeScale.interval;
        args.element.style.height = appHeight + 'px';
    }
    return (
        <ScheduleComponent width='100%' height='550px' ref={scheduleObj}
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
eventRendered={onEventRendered}>
            <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
    )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    EventRenderedArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const data: Object[] = [{
        Id: 13,
        Subject: 'Myths of Andromeda Galaxy',
        StartTime: new Date(2018, 1, 16, 10, 30),
        EndTime: new Date(2018, 1, 16, 10, 40)
    }, {
        Id: 14,
        Subject: 'Aliens vs Humans',
        StartTime: new Date(2018, 1, 15, 10, 0),
        EndTime: new Date(2018, 1, 15, 10, 20)
    }];
    const eventSettings: EventSettingsModel = { dataSource: data }
    const onEventRendered = (args: EventRenderedArgs): void => {
        let cellHeight: number = (scheduleObj.current.element.querySelector('.e-
work-cells' as HTMLElement).offsetHeight;
        let appHeight: number = ((args.data.EndTime as Date).getTime() -
(args.data.StartTime as Date).getTime()) / (60 * 1000) * (cellHeight *
scheduleObj.current.timeScale.slotCount) /
scheduleObj.current.timeScale.interval;
        args.element.style.height = appHeight + 'px';
    }
    return (

```



```

    <ScheduleComponent width='100%' height='550px' ref={scheduleObj}
    selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
    eventRendered={onEventRendered}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <link href="index.css" rel="stylesheet" />
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
        .e-past-app {
            background-color: chocolate !important;
        }
    </style>

```

```

        .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
        .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Block Dates and Times

It is possible to block a set of dates or a particular time ranges on the Scheduler. To do so, define an appointment object within `eventSettings` along with the required time range to block and set the `isBlock` field to true. Usually, the event objects defined with `isBlock` field set to true will block the entire time cells lying within the appropriate time ranges specified through `startTime` and `endTime` fields.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
'@syncfusion/ej2-react-schedule';
import { blockData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: blockData }
    return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { blockData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: blockData }
    return (
        <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings}>
            <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
    );
}

```

```

    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008c9f;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,

```

```

        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Block events can also be defined to repeat on several days as shown in the following code example.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, TimelineViews, Month, Agenda,
ViewsDirective, ViewDirective, Inject } from '@syncfusion/ej2-react-
schedule';
const App = () => {
    const data = [{
        Id: 1,
        Subject: 'Explosion of Betelgeuse Star',
        StartTime: new Date(2018, 1, 15, 9, 30),
        EndTime: new Date(2018, 1, 15, 11, 0),
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=5',
        IsBlock: true
    }, {
        Id: 2,
        Subject: 'Thule Air Crash Report',
        StartTime: new Date(2018, 1, 14, 12, 0),
        EndTime: new Date(2018, 1, 14, 14, 0)
    }];
    const eventSettings = { dataSource: data }
    return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='TimelineWeek' />
            <ViewDirective option='Month' />
            <ViewDirective option='Agenda' />
        </ViewsDirective>
        <Inject services={[Day, Week, TimelineViews, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```
import {
  ScheduleComponent, Day, Week, TimelineViews, Month, Agenda,
  ViewsDirective, ViewDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const data: Object[] = [{
    Id: 1,
    Subject: 'Explosion of Betelgeuse Star',
    StartTime: new Date(2018, 1, 15, 9, 30),
    EndTime: new Date(2018, 1, 15, 11, 0),
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=5',
    IsBlock: true
  }, {
    Id: 2,
    Subject: 'Thule Air Crash Report',
    StartTime: new Date(2018, 1, 14, 12, 0),
    EndTime: new Date(2018, 1, 14, 14, 0)
  }];
  const eventSettings: EventSettingsModel = { dataSource: data }
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)}
    eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='Month' />
        <ViewDirective option='Agenda' />
      </ViewsDirective>
      <Inject services={[Day, Week, TimelineViews, Month, Agenda]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Readonly

An interaction with the appointments of Scheduler can be enabled/disabled using the **readonly** property. With this property enabled, you can simply navigate between the Scheduler dates, views and can be able to view the appointment details in the quick info window. Most importantly, the users are not allowed to perform any CRUD actions on Scheduler, when this property is set to true. By default, it is set as **false**.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
'@syncfusion/ej2-react-schedule';

```

```
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData }
  return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)} readonly={true} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)}
    readonly={true} eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-navigations/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Make specific events readonly

There are scenarios where you need to restrict the CRUD action on specific appointments alone based on certain conditions. In the following example, the events that has occurred on the past hours from the current date of the Scheduler are made as read-only and the CRUD actions has been prevented only on those appointments. This can be achieved by setting `isReadOnly` field of read-only events to `true`.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
import { readOnlyData } from '../datasource';
const App = () => {
    const eventSettings = { dataSource: readOnlyData }

```



```

    return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
  }
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { readOnlyData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: readOnlyData }
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)}
    eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />

```

```

    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <link href="index.css" rel="stylesheet" />
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
        .e-past-app {
            background-color: chocolate !important;
        }
        .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
        .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

By default, the event editor is prevented to open on the read-only events when `isReadOnly` field is set to `true`.

Restricting event creation on specific time slots

You can restrict the users to create and update more than one appointment on specific time slots. Also, you can disable the CRUD action on those time slots if it is already occupied, which can be achieved using Scheduler's public method `isSlotAvailable`.

Note: The `isSlotAvailable` is centered around verifying appointments within the present view's date range. Yet, it does not encompass an evaluation of availability for recurrence occurrences that fall beyond this particular date range.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Inject } from
'@syncfusion/ej2-react-schedule';

```

```
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData }
  const onActionBegin = (args) => {
    if (args.requestType === 'eventCreate' && args.data.length > 0) {
      let eventData = args.data[0];
      let eventField = scheduleObj.current.eventFields;
      let startDate = eventData[eventField.startTime];
      let endDate = eventData[eventField.endTime];
      args.cancel = !scheduleObj.current.isSlotAvailable(startDate, endDate);
    }
  }
  return (<ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
actionBegin={onActionBegin}>
    <Inject services={[Day, Week, WorkWeek]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Inject,
  ActionEventArgs, EventFieldsMapping, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  const onActionBegin = (args: ActionEventArgs): void => {
    if (args.requestType === 'eventCreate' && (args.data as Object).length >
0) {
      let eventData: { [key: string]: Object } = args.data[0] as { [key:
string]: Object };
      let eventField: EventFieldsMapping = scheduleObj.current.eventFields;
      let startDate: Date = eventData[eventField.startTime] as Date;
      let endDate: Date = eventData[eventField.endTime] as Date;
      args.cancel = !scheduleObj.current.isSlotAvailable(startDate, endDate);
    }
  }
  return (
    <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
actionBegin={onActionBegin}>
      <Inject services={[Day, Week, WorkWeek]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
```

```
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
      background: green;
    }
  </style>
```

```

</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Differentiate the past time events

To differentiate the appearance of the appointments based on specific criteria such as displaying the past hour appointments with different colors on Scheduler, `eventRendered` event can be used which triggers before the appointment renders on the Scheduler.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef < ScheduleComponent > (null);
  const eventSettings = { dataSource: scheduleData }
  const onEventRendered = (args) => {
    if (args.data.EndTime < scheduleObj.current.selectedDate) {
      args.element.classList.add('e-past-app');
    }
  }
  return (<ScheduleComponent width='100%' height='550px' ref={scheduleObj}
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
eventRendered={onEventRendered}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventRenderedArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  const onEventRendered = (args: EventRenderedArgs): void => {
    if (args.data.EndTime < scheduleObj.current.selectedDate) {
      args.element.classList.add('e-past-app');
    }
  }

```

```

    }
  }
  return (
    <ScheduleComponent width='100%' height='550px' ref={scheduleObj}
    selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
    eventRendered={onEventRendered}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>

```

```

        .e-past-app {
            background-color: chocolate !important;
        }
        .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
        .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Appointments occupying entire cell

The Scheduler allows the event to occupies the full height of the cell without its header part by setting **true** for **enableMaxHeight** Property.

We can show more indicator if more than one appointment is available in a same cell by setting **true** to **enableIndicator** property whereas its default value is false.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, TimelineViews, TimelineMonth, ViewsDirective,
ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData, enableMaxHeight: true,
enableIndicator: false }
    return (<ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='TimelineWeek' />
            <ViewDirective option='TimelineMonth' />
        </ViewsDirective>
        <Inject services={[TimelineViews, TimelineMonth]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {

```

```

ScheduleComponent, TimelineViews, TimelineMonth, ViewsDirective,
ViewDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData,
enableMaxHeight: true, enableIndicator: false }
  return (
    <ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='TimelineMonth' />
      </ViewsDirective>
      <Inject services={[TimelineViews, TimelineMonth]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdn.jsdelivr.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
</style>

```



```

    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
      background: green;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

How to limit maximum number of events to display

In the Scheduler, the default behavior is to display concurrent events based on cell height, with each new event represented as

+n more characters. However, you may want to improve the quality of the presentation by limiting the number of concurrent events. This can be accomplished by using the [maxEventsPerRow](#) property, which is defaulted to the [views](#) property.

The [maxEventsPerRow](#) property is specific to the month, timeline month, and timeline year views, allowing you to view events visually in these rows. Below is a code example that demonstrates how to use this constraint and the events displayed in a cell have been created:

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Month, Inject, ViewsDirective, ViewDirective }
from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='380px' selectedDate={new
Date(2023, 11, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Month' maxEventsPerRow={3}/>
    </ViewsDirective>
    <Inject services={[Month]} />
  </ScheduleComponent>);
}

```

```
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Inject, Month,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='380px' selectedDate={new
Date(2023, 11, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Month' maxEventsPerRow={3} />
    </ViewsDirective>
    <Inject services={[Month]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
```

```

<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

The property [maxEventsPerRow](#) will be applicable only when [rowAutoHeight](#) feature is disabled in the Scheduler.

Display tooltip for appointments

The tooltip shows the Scheduler appointment's information in a formatted style by making use of the tooltip related options.

Show or hide built-in tooltip

The tooltip can be displayed for appointments by setting **true** to the **enableTooltip** option within the **eventSettings** property.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, TimelineViews, Month, Agenda, Inject }
from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData, enableTooltip: true }
    return (<ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, TimelineViews, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';

```

```
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, TimelineViews, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData,
    enableTooltip: true };
  return (
    <ScheduleComponent width='100%' height='500px' selectedDate={new
    Date(2018, 1, 15)} eventSettings={eventSettings}>
      <Inject services={[Day, Week, TimelineViews, Month, Agenda]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
```

```

        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Customizing event tooltip using template

After enabling the default tooltip, it is possible to customize the display of needed event information on tooltip by making use of the `tooltipTemplate` option within the `eventSettings`.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { eventsData } from './datasource';
const App = () => {
    const eventSettings = {
        dataSource: eventsData, enableTooltip: true,
        tooltipTemplate: template
    };
    const template = (props) => {
        return (<div className="tooltip-wrap">
            <div className="content-area"><div
className="name">{props.Subject}</div>
                {(props.City !== null && props.City !== undefined) ? <div
className="city">{props.City}</div> : ''}
                <div className="time">From : {props.StartTime.toLocaleString()}</div>
                <div className="time">To
: {props.EndTime.toLocaleString()}</div></div></div></div>);
    }
    return (<ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}

```

```
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { eventsData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = {
    dataSource: eventsData, enableTooltip: true,
    tooltipTemplate: template
  };
  const template = (props) => {
    return (<div className="tooltip-wrap">
      <div className="content-area"><div
className="name">{props.Subject}</div>
      {(props.City !== null && props.City !== undefined) ? <div
className="city">{props.City}</div> : ''}
      <div className="time">From : {props.StartTime.toLocaleString()}</div>
      <div className="time">To
: {props.EndTime.toLocaleString()}</div></div></div></div>);
    }
    return (
      <ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
      </ScheduleComponent>
    )
  };
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

All the field names that are mapped from the Scheduler dataSource to the appropriate field properties such as subject, description, location, startTime and endTime within the `eventSettings` can be accessed within the template.

Appointment selection

Appointment selection can be done either through mouse or keyboard actions. The selected events in UI will have a box shadow effect around to differentiate it from other appointments.

| Action | Description |

|-----|-----|

| Mouse click or Single tap on appointments | Selects single appointment. |

| Ctrl + [Mouse click] or [Single tap] on appointments | Selects multiple appointments. |

Deleting multiple appointments

With the options available to select multiple appointments, it is also possible to delete the multiple selected appointments simply by pressing the `delete` key. In case of deleting multiple selected occurrences of an event series, only those occurrences will be deleted and not the entire series.

Retrieve event details from the UI of an event

It is possible to access the information about the event fields of an appointment element displayed on the Scheduler UI. This can be achieved by passing an appointment element as argument to the public method `getEventDetails`.

In the following example, the subject of the appointment clicked has been displayed.

INDEX.JSX

```
import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef(null);
  const eventLog = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onEventClick = (args) => {
    let event = scheduleObj.current.getEventDetails(args.element);
    appendElement(event.Subject + '<hr>');
  }
  const appendElement = (html) => {
    let span = document.createElement('span');
    span.innerHTML = html;
    let log = eventLog.current;
    log.insertBefore(span, log.firstChild);
  }
  const onClick = () => {
    eventLog.current.innerHTML = '';
  }
  return (
    <div className='content-wrapper'>
      <div className='col-lg-9 control-section'>
        <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate=
          {new Date(2018, 1, 15)} eventSettings={eventSettings}
eventClick={onEventClick}>
          <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
      </div>
      <div className='col-lg-3 property-section'>
        <div title='Event Trace'>
          <table id='property' title='Properties' className='property-panel-
table' style={{ width: '100%' }}>
            <tbody>
              <tr style={{ height: '250px' }}>
                <td>
                  <div className='eventarea' style={{ height: '245px',
overflow: 'auto' }}>
                    <span className='EventLog' ref={eventLog} style={{
wordBreak: 'normal' }}></span>
                  </div>
                </td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  )
}
```



```

        </tr>
        <tr style={{ height: '50px' }}>
          <td style={{ width: '30%' }}>
            <div className='evtbtn' style={{ paddingBottom: '10px' }}>
              <ButtonComponent title='Clear'
onClick={onClick}>Clear</ButtonComponent>
            </div>
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
</div>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventClickArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventLog = useRef(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const onEventClick = (args: EventClickArgs): void => {
    let event: object = scheduleObj.current.getEventDetails(args.element as
HTMLElement);
    appendElement(event.Subject + '<hr>');
  }
  const appendElement = (html: string): void => {
    let span: HTMLElement = document.createElement('span');
    span.innerHTML = html;
    let log: HTMLElement = eventLog.current;
    log.insertBefore(span, log.firstChild);
  }
  const onClick = (): void => {
    eventLog.current.innerHTML = '';
  }
  return (
    <div className='content-wrapper'>
      <div className='col-lg-9 control-section'>
        <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate=
      {new Date(2018, 1, 15)} eventSettings={eventSettings}
eventClick={onEventClick}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
      </div>
    </div>
  );
};

```

```

        </ScheduleComponent>
      </div>
      <div className='col-lg-3 property-section'>
        <div title='Event Trace'>
          <table id='property' title='Properties' className='property-panel-table' style={{ width: '100%' }}>
            <tbody>
              <tr style={{ height: '250px' }}>
                <td>
                  <div className='eventarea' style={{ height: '245px', overflow: 'auto' }}>
                    <span className='EventLog' ref={eventLog} style={{ wordBreak: 'normal' }}></span>
                  </div>
                </td>
              </tr>
              <tr style={{ height: '50px' }}>
                <td style={{ width: '30%' }}>
                  <div className='evtbtn' style={{ paddingBottom: '10px' }}>
                    <ButtonComponent title='Clear'
                    onClick={onClick}>Clear</ButtonComponent>
                  </div>
                </td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-navigations/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>

    <div>
        <div id="schedule">
            <div id='loader'>Loading....</div>
        </div>
    </div>

</body>
</html>

```

Get the current view appointments

To retrieve the appointments present in the current view of the Scheduler, you can make use of the `getCurrentViewEvents` public method. In the following example, the count of current view appointment collection rendered has been traced in `dataBound` event.

INDEX.JSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
EventSettingsModel } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const scheduleObj = useRef(null);
    const eventLog = useRef(null);
    const eventSettings = { dataSource: scheduleData };
    const onDataBound = () => {
        let event = scheduleObj.current.getCurrentViewEvents();
        if (event.length > 0) {

```

```

    appendElement('Events present on current view <b>' + event.length +
    '<b><hr>');
    } else {
        appendElement('No Events available in this view.<hr>');
    }
}
const appendElement = (html) => {
    let span = document.createElement('span');
    span.innerHTML = html;
    let log = eventLog.current;
    log.insertBefore(span, log.firstChild);
}
const onClick = () => {
    eventLog.current.innerHTML = '';
}
return (<div className='content-wrapper'>
    <div className='col-lg-9 control-section'>
        <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate=
        {new Date(2018, 1, 15)} eventSettings={eventSettings}
dataBound={onDataBound}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
    </div>
    <div className='col-lg-3 property-section'>
        <div title='Event Trace'>
            <table id='property' title='Properties' className='property-panel-
table' style={{ width: '100%' }}>
                <tbody>
                    <tr style={{ height: '250px' }}>
                        <td>
                            <div className='eventarea' style={{ height: '245px',
overflow: 'auto' }}>
                                <span className='EventLog' ref={eventLog} style={{
wordBreak: 'normal' }}></span>
                            </div>
                        </td>
                    </tr>
                    <tr style={{ height: '50px' }}>
                        <td style={{ width: '30%' }}>
                            <div className='evtbtn' style={{ paddingBottom: '10px' }}>
                                <ButtonComponent title='Clear'
onClick={onClick}>Clear</ButtonComponent>
                            </div>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
    </div>
</div>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const eventLog = useRef(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onDataBound = (): void => {
        let event: Object[] = scheduleObj.current.getCurrentViewEvents();
        if (event.length > 0) {
            appendElement('Events present on current view <b>' + event.length +
                '<b><hr>');
        } else {
            appendElement('No Events available in this view.<hr>');
        }
    }
    const appendElement = (html: string): void => {
        let span: HTMLElement = document.createElement('span');
        span.innerHTML = html;
        let log: HTMLElement = eventLog.current;
        log.insertBefore(span, log.firstChild);
    }
    const onClick = (): void => {
        eventLog.current.innerHTML = '';
    }
    return (<div className='content-wrapper'>
        <div className='col-lg-9 control-section'>
            <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
                selectedDate=
                    {new Date(2018, 1, 15)} eventSettings={eventSettings}
                dataBound={onDataBound}>
                <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
            </ScheduleComponent>
        </div>
        <div className='col-lg-3 property-section'>
            <div title='Event Trace'>
                <table id='property' title='Properties' className='property-panel-
                    table' style={{ width: '100%' }}>
                    <tbody>
                        <tr style={{ height: '250px' }}>
                            <td>
                                <div className='eventarea' style={{ height: '245px',
                                    overflow: 'auto' }}>
                                    <span className='EventLog' ref={eventLog} style={{
                                        wordBreak: 'normal' }}></span>
                                </div>
                            </td>
                        </tr>
                        <tr style={{ height: '50px' }}>
                            <td style={{ width: '30%' }}>
                                <div className='evtbtn' style={{ paddingBottom: '10px' }}>

```

```

        <ButtonComponent title='Clear'
onClick={onClick}>Clear</ButtonComponent>
      </div>
    </td>
  </tr>
</tbody>
</table>
</div>
</div>
</div>)
</div>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;

```

```

    }
  </style>
</head>
<body>

    <div>
      <div id="schedule">
        <div id='loader'>Loading....</div>
      </div>
    </div>

</body>
</html>

```

Get the entire appointment collections

The entire collection of appointments rendered on the Scheduler can be accessed using the `getEvents` public method. In the following example, the count of entire appointment collection rendered on the Scheduler has been traced in `dataBound` event.

INDEX.JSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef(null);
  const eventLog = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onDataBound = () => {
    let event = scheduleObj.current.getEvents();
    appendElement('Events present on scheduler <b>' + event.length +
    '<b><hr>');
  }
  const appendElement = (html) => {
    let span = document.createElement('span');
    span.innerHTML = html;
    let log = eventLog.current;
    log.insertBefore(span, log.firstChild);
  }
  const onClick = () => {
    eventLog.current.innerHTML = '';
  }
  return (
    <div className='content-wrapper'>
      <div className='col-lg-9 control-section'>
        <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate=
      {new Date(2018, 1, 15)} eventSettings={eventSettings}
dataBound={onDataBound}>
          <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
      </div>
    </div>

```

```

        <div className='col-lg-3 property-section'>
            <div title='Event Trace'>
                <table id='property' title='Properties' className='property-panel-
table' style={{ width: '100%' }}>
                    <tbody>
                        <tr style={{ height: '250px' }}>
                            <td>
                                <div className='eventarea' style={{ height: '245px',
overflow: 'auto' }}>
                                    <span className='EventLog' ref={eventLog} style={{
wordBreak: 'normal' }}></span>
                                </div>
                            </td>
                        </tr>
                        <tr style={{ height: '50px' }}>
                            <td style={{ width: '30%' }}>
                                <div className='evtbtn' style={{ paddingBottom: '10px' }}>
                                    <ButtonComponent title='Clear'
onClick={onClick}>Clear</ButtonComponent>
                                </div>
                            </td>
                        </tr>
                    </tbody>
                </table>
            </div>
        </div>
    </div>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const eventLog = useRef(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onDataBound = (): void => {
        let event: Object[] = scheduleObj.current.getEvents();
        appendElement('Events present on scheduler <b>' + event.length +
'<b><hr>');
    }
    const appendElement = (html: string): void => {
        let span: HTMLElement = document.createElement('span');
        span.innerHTML = html;
        let log: HTMLElement = eventLog.current;
    }

```



```

    log.insertBefore(span, log.firstChild);
  }
  const onClick = (): void => {
    eventLog.current.innerHTML = '';
  }
  return (
    <div className='content-wrapper'>
      <div className='col-lg-9 control-section'>
        <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate=
        {new Date(2018, 1, 15)} eventSettings={eventSettings}
dataBound={onDataBound}>
          <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
      </div>
      <div className='col-lg-3 property-section'>
        <div title='Event Trace'>
          <table id='property' title='Properties' className='property-panel-
table' style={{ width: '100%' }}>
            <tbody>
              <tr style={{ height: '250px' }}>
                <td>
                  <div className='eventarea' style={{ height: '245px',
overflow: 'auto' }}>
                    <span className='EventLog' ref={eventLog} style={{
wordBreak: 'normal' }}></span>
                  </div>
                </td>
                <tr style={{ height: '50px' }}>
                  <td style={{ width: '30%' }}>
                    <div className='evtbtn' style={{ paddingBottom: '10px' }}>
                      <ButtonComponent title='Clear'
onClick={onClick}>Clear</ButtonComponent>
                    </div>
                  </td>
                </tr>
              </tbody>
            </table>
          </div>
        </div>
      </div>
    </div>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />

```

```

    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <link href="index.css" rel="stylesheet" />
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>
<body>

        <div>
            <div id="schedule">
                <div id='loader'>Loading....</div>
            </div>
        </div>

</body>
</html>

```

Refresh appointments

If your requirement is to simply refresh the appointments instead of refreshing the entire Scheduler elements from your application end, make use of the `refreshEvents` public method.

```
`ts
```

```
this.scheduleObj.refreshEvents();
```

```
,
```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Data binding in React Schedule component

The Scheduler uses **DataManager**, which supports both RESTful data service binding and JavaScript object array binding. The [dataSource](#) property of Scheduler can be assigned either with the instance of **DataManager** or JavaScript object array collection, as it supports the following two kind of data binding methods:

- Local data
- Remote data

Binding local data

To bind local JSON data to the Scheduler, you can simply assign a JavaScript object array to the [dataSource](#) option of the scheduler within the **eventSettings** property. The JSON object **dataSource** can also be provided as an instance of **DataManager** and assigned to the Scheduler **dataSource** property.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
const App = () => {
  let data = [{
    Id: 1,
    Subject: 'Explosion of Betelgeuse Star',
    StartTime: new Date(2018, 1, 15, 9, 30),
    EndTime: new Date(2018, 1, 15, 11, 0)
  }, {
    Id: 2,
    Subject: 'Thule Air Crash Report',
    StartTime: new Date(2018, 1, 12, 12, 0),
    EndTime: new Date(2018, 1, 12, 14, 0)
  }, {
    Id: 3,
    Subject: 'Blue Moon Eclipse',
    StartTime: new Date(2018, 1, 13, 9, 30),
    EndTime: new Date(2018, 1, 13, 11, 0)
  }, {
    Id: 4,
    Subject: 'Meteor Showers in 2018',
    StartTime: new Date(2018, 1, 14, 13, 0),
    EndTime: new Date(2018, 1, 14, 14, 30)
  }
  ];
  const eventSettings = { dataSource: data };
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
EventSettingsModel } from '@syncfusion/ej2-react-schedule';
const App = () => {
  let data: Object[] = [{
    Id: 1,
    Subject: 'Explosion of Betelgeuse Star',
    StartTime: new Date(2018, 1, 15, 9, 30),
    EndTime: new Date(2018, 1, 15, 11, 0)
  }, {
    Id: 2,
    Subject: 'Thule Air Crash Report',
    StartTime: new Date(2018, 1, 12, 12, 0),
    EndTime: new Date(2018, 1, 12, 14, 0)
  }, {
    Id: 3,
    Subject: 'Blue Moon Eclipse',
    StartTime: new Date(2018, 1, 13, 9, 30),
    EndTime: new Date(2018, 1, 13, 11, 0)
  }, {
    Id: 4,
    Subject: 'Meteor Showers in 2018',
    StartTime: new Date(2018, 1, 14, 13, 0),
    EndTime: new Date(2018, 1, 14, 14, 30)
  }
  ];
  const eventSettings: EventSettingsModel = { dataSource: data };
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)}
      eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

By default, **DataManager** uses **JsonAdaptor** for binding local data.

You can also bind different field names to the default event fields as well as include additional custom fields to the event object collection which can be referred [here](#).

Binding remote data

Any kind of remote data services can be bound to the Scheduler. To do so, create an instance of **DataManager** and provide the service URL to the **url** option of **DataManager** and then assign it to the **dataSource** property within **eventSettings**.

Using ODataV4Adaptor

ODataV4 is a standardized protocol for creating and consuming data. Refer to the following code example to retrieve the data from ODataV4 service using the **DataManager**. To connect with ODataV4 service end points, it is necessary to make use of **ODataV4Adaptor** within **DataManager**.

INDEX.JSX

```

import { useState, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```

import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
const App = () => {
  const [dataManager, setDataManager] = useState(null);
  useEffect(() => {
    const fetchData = async () => {
      const manager = new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
        adaptor: new ODataV4Adaptor()
      });
      await manager.ready;
      setDataManager(manager);
    };
    fetchData();
  }, []);
  const fieldsData = {
    id: 'Id',
    subject: { name: 'ShipName' },
    location: { name: 'ShipCountry' },
    description: { name: 'ShipAddress' },
    startTime: { name: 'OrderDate' },
    endTime: { name: 'RequiredDate' },
    recurrenceRule: { name: 'ShipRegion' }
  }
  const eventSettings = { dataSource: dataManager, fields: fieldsData };
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(1996, 6, 9)}
    readonly={true} eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  );
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useState, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
import { DataManager, ODataV4Adaptor, EventSettingsModel } from
 '@syncfusion/ej2-data';
const App = () => {
  const [dataManager, setDataManager] = useState(null);
  useEffect(() => {
    const fetchData = async () => {
      const manager = new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
        adaptor: new ODataV4Adaptor()
      });
      await manager.ready;
      setDataManager(manager);
    };
  });
}

```

```

    fetchData();
  }, []);
  const fieldsData = {
    id: 'Id',
    subject: { name: 'ShipName' },
    location: { name: 'ShipCountry' },
    description: { name: 'ShipAddress' },
    startTime: { name: 'OrderDate' },
    endTime: { name: 'RequiredDate' },
    recurrenceRule: { name: 'ShipRegion' }
  }
  const eventSettings: EventSettingsModel = { dataSource: dataManager,
fields: fieldsData };

  return (
    <ScheduleComponent height='550px' selectedDate={new Date(1996, 6, 9)}
    readonly={true} eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  );
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>

```

```

<script src="systemjs.config.js"></script>
<style>
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Filter events using the in-built query

To enable server-side filtering operations based on predetermined conditions, the [includeFiltersInQuery](#) API can be set to true, this allows the filter query to be constructed using the start date, end date, and recurrence rule which in turn enables the request to be filtered accordingly.

This method greatly improves the component's performance by reducing the data that needs to be transferred to the client side. As a result, the component's efficiency and responsiveness are significantly enhanced, resulting in a better user experience. However, it is important to consider the possibility of longer query strings, which may cause issues with the maximum URL length or server limitations on query string length.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
const App = () => {
  let dataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
    adaptor: new ODataV4Adaptor()
  });
  const fieldsData = {
    id: 'Id',
    subject: { name: 'ShipName' },
    location: { name: 'ShipCountry' },
    description: { name: 'ShipAddress' },
    startTime: { name: 'OrderDate' },
    endTime: { name: 'RequiredDate' },
    recurrenceRule: { name: 'ShipRegion' }
  }
  const eventSettings = { includeFiltersInQuery: true, dataSource:
dataManager, fields: fieldsData }

```



```

    return (<ScheduleComponent height='550px' currentView='Month'
selectedDate={new Date(1996, 6, 9)} readonly={true}
eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
</ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
EventSettingsModel } from '@syncfusion/ej2-react-schedule';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
const App = () => {
    let dataManager: DataManager = new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
        adaptor: new ODataV4Adaptor()
    });
    const fieldsData = {
        id: 'Id',
        subject: { name: 'ShipName' },
        location: { name: 'ShipCountry' },
        description: { name: 'ShipAddress' },
        startTime: { name: 'OrderDate' },
        endTime: { name: 'RequiredDate' },
        recurrenceRule: { name: 'ShipRegion' }
    }
    const eventSettings: EventSettingsModel = { includeFiltersInQuery: true,
dataSource: dataManager, fields: fieldsData }
    return (
        <ScheduleComponent height='550px' currentView='Month' selectedDate={new
Date(1996, 6, 9)} readonly={true}
        eventSettings={eventSettings}>
            <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
    )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />


```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

The following image represents how the parameters are passed using ODataV4 filter.

×	Headers	Payload	Preview	Response	Initiator	Timing
▼ General						
Request URL: https://services.odata.org/V4/Northwind/Northwind.svc/Orders/?\$filter=(((((OrderDate%20ge%201996-06-29T18:30:00.000Z)%20and%20(RequiredDate%20ge%201996-06-29T18:30:00.000Z))%20and%20(OrderDate%20lt%201996-08-03T18:30:00.000Z))%20or%20((OrderDate%20le%201996-06-29T18:30:00.000Z)%20and%20(RequiredDate%20gt%201996-06-29T18:30:00.000Z)))%20or%20((ShipRegion%20ne%20null)%20and%20(ShipRegion%20ne%20%27%27))						
Request Method: GET						
Status Code:  200 OK						
Remote Address: 137.117.17.70:443						
Referrer Policy: strict-origin-when-cross-origin						

Using custom adaptor

It is possible to create your own custom adaptor by extending the built-in available adaptors. The following example demonstrates the custom adaptor usage and how to add a custom field `EventID` for the appointments by overriding the built-in response processing using the `processResponse` method of the `ODataV4Adaptor`.

INDEX.JSX

```
import { useState, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
const CustomAdaptor = () => {
    ODataV4Adaptor.call(this);
}
CustomAdaptor.prototype = Object.create(ODataV4Adaptor.prototype);
CustomAdaptor.prototype.constructor = CustomAdaptor;
CustomAdaptor.prototype.processResponse = function () {
    let i = 0;
    let original = ODataV4Adaptor.prototype.processResponse.apply(this, arguments);
    original.forEach((item) => (item['EventID'] = ++i));
    return original;
};
const App = () => {
    const [dataManager, setDataManager] = useState(null);
    useEffect(() => {
        const fetchData = async () => {
            const manager = new DataManager({
```

```

        url:
        'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
        adaptor: new CustomAdaptor(),
    });
    const query = new Query().take(10); // Example query to limit the
number of records
    const data = await manager.executeQuery(query);
    setDataManager(new DataManager(data));
    };
    fetchData();
}, []);
const fieldsData = {
    id: 'Id',
    subject: { name: 'ShipName' },
    location: { name: 'ShipCountry' },
    description: { name: 'ShipAddress' },
    startTime: { name: 'OrderDate' },
    endTime: { name: 'RequiredDate' },
    recurrenceRule: { name: 'ShipRegion' }
}
const eventSettings = { dataSource: dataManager, fields: fieldsData };
return (
    <ScheduleComponent
        height='550px'
        selectedDate={new Date(1996, 6, 9)}
        readonly={true}
        eventSettings={eventSettings}
    >
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
);
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useState, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
EventSettingsModel } from '@syncfusion/ej2-react-schedule';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
function CustomAdaptor() {
    ODataV4Adaptor.call(this);
}
CustomAdaptor.prototype = Object.create(ODataV4Adaptor.prototype);
CustomAdaptor.prototype.constructor = CustomAdaptor;
CustomAdaptor.prototype.processResponse = function () {
    let i = 0;
    let original = ODataV4Adaptor.prototype.processResponse.apply(this,
arguments);
    original.forEach((item: any) => (item['EventID'] = ++i));
    return original;
};
const App = () => {

```

```

const [dataManager, setDataManager] = useState<DataManager | null>(null);
useEffect(() => {
    const fetchData = async () => {
        const manager = new DataManager({
            url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
            adaptor: new CustomAdaptor(),
            crossDomain: true
        });
        await manager.ready;
        setDataManager(manager);
    };
    fetchData();
}, []);

const fieldsData = {
    id: 'Id',
    subject: { name: 'ShipName' },
    location: { name: 'ShipCountry' },
    description: { name: 'ShipAddress' },
    startTime: { name: 'OrderDate' },
    endTime: { name: 'RequiredDate' },
    recurrenceRule: { name: 'ShipRegion' }
}

const eventSettings: EventSettingsModel = { dataSource: dataManager,
fields: fieldsData };
return (
    <ScheduleComponent
        height='550px'
        selectedDate={new Date(1996, 6, 9)}
        readonly={true}
        eventSettings={eventSettings}
    >
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
);
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Loading data via AJAX post

You can bind the event data through external ajax request and assign it to the `dataSource` property of Scheduler. In the following code example, we have retrieved the data from server with the help of ajax request and assigned the resultant data to the `dataSource` property of Scheduler within the `onSuccess` event of Ajax.

[src/app/app.tsx]

```
`ts
```

```
import * as React from 'react';
```

```
import { useState, useEffect } from 'react';
```

```
import ReactDOM from 'react-dom';
```

```
import { Ajax } from '@syncfusion/ej2-base';
```

```
import {
```

```
ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject, EventSettingsModel
```

```

} from '@syncfusion/ej2-react-schedule';
import { DataManager } from '@syncfusion/ej2-data';
const App = () => {
  const [dataManager, setDataManager] = useState<DataManager | null>(null);
  useEffect(() => {
    const ajax = new Ajax('Home/GetData', 'GET', false);
    ajax.send();
    ajax.onSuccess = function (value: DataManager) {
      setDataManager(value);
    };
  }, []);
  const eventSettings: EventSettingsModel = { dataSource: dataManager };
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(2017, 5, 11)}
      eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  );
}

const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

[src/app/app.ts]

Definition for the controller method `GetData` can be referred [here](#).

Passing additional parameters to the server

To send an additional custom parameter to the server-side post, you need to make use of the `addParams` method of `Query`. Now, assign this `Query` object with additional parameters to the `query` property of Scheduler.

INDEX.JSX

```

import { useState, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
const App = () => {
  const [dataManager, setDataManager] = useState(null);

```

```

useEffect(() => {
    const fetchData = async () => {
        const manager = new DataManager({
            url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
            adaptor: new ODataV4Adaptor()
        });
        await manager.ready;
        const query = new Query().addParams('readOnly', 'true');
        const data = await manager.executeQuery(query);
        setDataManager(manager);
    };
    fetchData();
}, []);
const fieldsData = {
    id: 'Id',
    subject: { name: 'ShipName' },
    location: { name: 'ShipCountry' },
    description: { name: 'ShipAddress' },
    startTime: { name: 'OrderDate' },
    endTime: { name: 'RequiredDate' },
    recurrenceRule: { name: 'ShipRegion' }
}
const eventSettings = { dataSource: dataManager, fields: fieldsData };
return (
    <ScheduleComponent
        height='550px'
        readonly={true}
        eventSettings={eventSettings}
        selectedDate={new Date(1996, 6, 9)}
    >
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
);
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useState, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
EventSettingsModel } from '@syncfusion/ej2-react-schedule';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
const App = () => {
    const [dataManager, setDataManager] = useState<DataManager | null>(null);
    useEffect(() => {
        const fetchData = async () => {
            const manager = new DataManager({
                url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
                adaptor: new ODataV4Adaptor()
            });
            await manager.ready;
            const query = new Query().addParams('readOnly', 'true');
            const data = await manager.executeQuery(query) as any;

```



```

        setDataManager(manager);
    };
    fetchData();
}, []);
const fieldsData = {
    id: 'Id',
    subject: { name: 'ShipName' },
    location: { name: 'ShipCountry' },
    description: { name: 'ShipAddress' },
    startTime: { name: 'OrderDate' },
    endTime: { name: 'RequiredDate' },
    recurrenceRule: { name: 'ShipRegion' }
}
const eventSettings: EventSettingsModel = { dataSource: dataManager,
fields: fieldsData };
return (
    <ScheduleComponent
        height='550px'
        readonly={true}
        eventSettings={eventSettings}
        selectedDate={new Date(1996, 6, 9)}
    >
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
);
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />

```

```

    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

The parameters added using the [query](#) property will be sent along with the data request sent to the server on every scheduler actions.

Handling failure actions

During the time of Scheduler interacting with server, there are chances that some server-side exceptions may occur. You can acquire those error messages or exception details in client-side using the [actionFailure](#) event of Scheduler.

The argument passed to the [actionFailure](#) event contains the error details returned from the server.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { DataManager } from '@syncfusion/ej2-data';
const App = () => {
    const scheduleRef = useRef(null);
    const dataManager = useRef(new DataManager({
        url: 'http://some.com/invalidUrl'
    }));
    const eventSettings = { dataSource: dataManager.current };
    const onActionFailure = () => {
        const span = document.createElement('span');
        scheduleRef.current?.element.parentNode?.insertBefore(span,
scheduleRef.current?.element);
        if (span.style) {
            span.style.color = '#FF0000';
        }
    }
}

```

```

        span.innerHTML = 'Server exception: 404 Not found';
    };
    return (
        <ScheduleComponent height='550px' ref={scheduleRef} selectedDate={new
Date(2017, 5, 11)} actionFailure={onActionFailure}
eventSettings={eventSettings} >
            <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
    );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
EventSettingsModel } from '@syncfusion/ej2-react-schedule';
import { DataManager } from '@syncfusion/ej2-data';
const App = () => {
    const scheduleRef = useRef<ScheduleComponent>(null);
    const dataManager = useRef<DataManager>(new DataManager({
        url: 'http://some.com/invalidUrl'
    }));
    const eventSettings: EventSettingsModel = { dataSource: dataManager.current
};
    const onActionFailure = (): void => {
        const span = document.createElement('span');
        scheduleRef.current.element.parentNode.insertBefore(span,
scheduleRef.current.element);
        if (span.style) {
            span.style.color = '#FF0000';
        }
        span.innerHTML = 'Server exception: 404 Not found';
    };
    return (
        <ScheduleComponent height='550px' ref={scheduleRef} selectedDate={new
Date(2017, 5, 11)} actionFailure={onActionFailure}
eventSettings={eventSettings} >
            <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
    );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

```

```

<meta name="description" content="Essential JS 2 for React Components" />
<meta name="author" content="Syncfusion" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

The [actionFailure](#) event will be triggered not only on server returning errors, but also when there is an exception while processing any of the Scheduler CRUD actions.

Scheduler CRUD actions

The CRUD (Create, Read, Update and Delete) actions can be performed easily on Scheduler appointments using the various adaptors available within the **DataManager**. Most preferably, we will be using **UrlAdaptor** for performing CRUD actions on scheduler appointments.

```
`ts
```

```
import { Schedule, Day, Week, WorkWeek, Month, Agenda } from '@syncfusion/ej2-schedule';
```

```
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
```

```

Schedule.Inject(Day, Week, WorkWeek, Month, Agenda);
let dataManager: DataManager = new DataManager({
url: 'Home/GetData', // 'controller/actions'
crudUrl: 'Home/UpdateData',
adaptor: new UrlAdaptor
});
let scheduleObj: Schedule = new Schedule({
height: '550px',
selectedDate: new Date(2017, 5, 5),
eventSettings: { dataSource: dataManager }
});
scheduleObj.appendTo('#Schedule');
`

```

The server-side controller code to handle the CRUD operations are as follows.

```

`c#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using ScheduleSample.Models;
namespace ScheduleSample.Controllers
{
public class HomeController : Controller
{
ScheduleDataDataContext db = new ScheduleDataDataContext();
public ActionResult Index()
{
return View();
}

public JsonResult LoadData() // Here we get the Start and End Date and based on that can filter the data
and return to Scheduler
{

```

```
var data = db.ScheduleEventDatas.ToList();
return Json(data, JsonRequestBehavior.AllowGet);
}
[HttpPost]
public JsonResult UpdateData(EditParams param)
{
    if (param.action == "insert" || (param.action == "batch" && param.added != null)) // this block of code
        will execute while inserting the appointments
    {
        var value = (param.action == "insert") ? param.value : param.added[0];
        int intMax = db.ScheduleEventDatas.ToList().Count > 0 ? db.ScheduleEventDatas.ToList().Max(p => p.Id) :
        1;
        DateTime startTime = Convert.ToDateTime(value.StartTime);
        DateTime endTime = Convert.ToDateTime(value.EndTime);
        ScheduleEventData appointment = new ScheduleEventData()
        {
            Id = intMax + 1,
            StartTime = startTime.ToLocalTime(),
            EndTime = endTime.ToLocalTime(),
            Subject = value.Subject,
            IsAllDay = value.IsAllDay,
            StartTimezone = value.StartTimezone,
            EndTimezone = value.EndTimezone,
            RecurrenceRule = value.RecurrenceRule,
            RecurrenceID = value.RecurrenceID,
            RecurrenceException = value.RecurrenceException
        };
        db.ScheduleEventDatas.InsertOnSubmit(appointment);
        db.SubmitChanges();
    }
    if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of
        code will execute while updating the appointment
    {
        var value = (param.action == "update") ? param.value : param.changed[0];
```

```
var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
if (filterData.Count() > 0)
{
    DateTime startTime = Convert.ToDateTime(value.StartTime);
    DateTime endTime = Convert.ToDateTime(value.EndTime);
    ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
    Convert.ToInt32(value.Id));
    appointment.StartTime = startTime.ToLocalTime();
    appointment.EndTime = endTime.ToLocalTime();
    appointment.StartTimezone = value.StartTimezone;
    appointment.EndTimezone = value.EndTimezone;
    appointment.Subject = value.Subject;
    appointment.IsAllDay = value.IsAllDay;
    appointment.RecurrenceRule = value.RecurrenceRule;
    appointment.RecurrenceID = value.RecurrenceID;
    appointment.RecurrenceException = value.RecurrenceException;
}
db.SubmitChanges();
}

if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of
code will execute while removing the appointment
{
    if (param.action == "remove")
    {
        int key = Convert.ToInt32(param.key);
        ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
        if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
    }
    else
    {
        foreach (var apps in param.deleted)
        {
            ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
            if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
        }
    }
}
```

```

}
}
db.SubmitChanges();
}
var data = db.ScheduleEventDatas.ToList();
return Json(data, JsonRequestBehavior.AllowGet);
}
public class EditParams
{
    public string key { get; set; }
    public string action { get; set; }
    public List<ScheduleEventData> added { get; set; }
    public List<ScheduleEventData> changed { get; set; }
    public List<ScheduleEventData> deleted { get; set; }
    public ScheduleEventData value { get; set; }
}
}
}
,

```

Configuring Scheduler with Google API service

We have assigned our custom created Google Calendar url to the DataManager and assigned the same to the Scheduler `dataSource`. Since the events data retrieved from the Google Calendar will be in its own object format, therefore it needs to be resolved manually within the Scheduler's `dataBinding` event. Within this event, the event fields needs to be mapped properly and then assigned to the result.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
const App = () => {
    let calendarId = 'en.usa%23holiday@group.v.calendar.google.com';
    let publicKey = 'AIzaSyBgbX_tgmVanBP4yafDPPXxWr70sjbKAXM';
    const dataManger = new DataManager({
        url: 'https://www.googleapis.com/calendar/v3/calendars/' + calendarId
        + '/events?key=' + publicKey,
        adaptor: new WebApiAdaptor(),
        crossDomain: true
    });
    const eventSettings = { dataSource: dataManger };
    const onDataBinding = (e) => {

```



```

        let items = e.result.items;
        let scheduleData = [];
        if (items.length > 0) {
            for (let i = 0; i < items.length; i++) {
                let event = items[i];
                let when = event.start.dateTime;
                let start = event.start.dateTime;
                let end = event.end.dateTime;
                if (!when) {
                    when = event.start.date;
                    start = event.start.date;
                    end = event.end.date;
                }
                scheduleData.push({
                    Id: event.id,
                    Subject: event.summary,
                    StartTime: new Date(start),
                    EndTime: new Date(end),
                    IsAllDay: !event.start.dateTime
                });
            }
        }
        e.result = scheduleData;
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 10, 14)} readonly={true} eventSettings={eventSettings}
dataBinding={onDataBinding}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
EventSettingsModel } from '@syncfusion/ej2-react-schedule';
import { DataManager, WebApiAdaptor, Query } from '@syncfusion/ej2-data';
const App = () => {
    let calendarId: string = 'en.usa%23holiday@group.v.calendar.google.com';
    let publicKey: 'AIzaSyBgbX_tgmVanBP4yafDPPXxWr70sjsbKAXM';
    const dataManger: DataManager = new DataManager({
        url: 'https://www.googleapis.com/calendar/v3/calendars/' + calendarId +
        '/events?key=' + publicKey,
        adaptor: new WebApiAdaptor(),
        crossDomain: true
    });
    const eventSettings: EventSettingsModel = { dataSource: dataManger };
    const onDataBinding = (e: { [key: string]: Object }): void => {
        let items: { [key: string]: Object }[] = (e.result as { [key: string]:
Object }).items as { [key: string]: Object }[];
        let scheduleData: Object[] = [];
        if (items.length > 0) {

```

```

    for (let i: number = 0; i < items.length; i++) {
        let event: { [key: string]: Object } = items[i];
        let when: string = (event.start as { [key: string]: Object
    }).dateTime as string;
        let start: string = (event.start as { [key: string]: Object
    }).dateTime as string;
        let end: string = (event.end as { [key: string]: Object }).dateTime
as string;
        if (!when) {
            when = (event.start as { [key: string]: Object }).date as string;
            start = (event.start as { [key: string]: Object }).date as string;
            end = (event.end as { [key: string]: Object }).date as string;
        }
        scheduleData.push({
            Id: event.id,
            Subject: event.summary,
            StartTime: new Date(start),
            EndTime: new Date(end),
            IsAllDay: !(event.start as { [key: string]: Object }).dateTime
        });
    }
    e.result = scheduleData;
}
return (
    <ScheduleComponent width='100%'
        height='550px' selectedDate={new Date(2018, 10, 14)} readonly={true}
        eventSettings={eventSettings} dataBinding={onDataBinding}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
)
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Crud actions in React Schedule component

Events, a.k.a. Appointments, play an important role in Scheduler with which the users mostly interact. You can easily manipulate (add/edit/delete) the desired appointments as and when required either using the editor window or through the drag and resize action.

Add

Any kind of appointments such as normal, all-day, spanned or recurring events can be easily added on Scheduler using any one of the following ways.

- [Creation using editor window](#)
- [Creation Using addEvent method](#)

Creation using editor window

The default editor window opens when you double click on the Scheduler cells. It provides you with event related options such as Subject, Location, Start and End time, All-day, Timezone, Description and other recurrence options. With these available fields, you can choose to provide detailed information to the events. Once the fields are filled with proper values, enter the **Save** button to add an event.

In case, if you want to simply provide the Subject alone for appointments, just single click on the required cells which will open the quick popup expecting you to enter subject alone and save it. You can also select multiple cells and press **Enter** key to open the quick popup for selected time range and save the appointment for that time range.

In case, if you need to add some other additional fields to the editor window, then you can opt for [custom editor window](#) which allows you to include fields as per your application needs. If you need to add just one or two [additional fields to the existing default editor window](#), you can do so by defining it manually and then appending it to the editor window.

Creation using addEvent method

The appointments can be created dynamically by using **addEvent** method. Either you can add a single or a collection of appointment objects using **addEvent** method. The following code example let you know how to use the **addEvent** method to create multiple appointments simultaneously.

INDEX.JSX

```
import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef(null);
  const buttonObj = useRef(null);
  let scheduleData = [{
    Id: 3,
    Subject: 'Testing',
    StartTime: new Date(2018, 1, 11, 9, 0),
    EndTime: new Date(2018, 1, 11, 10, 0),
    IsAllDay: false
  }, {
    Id: 4,
    Subject: 'Vacation',
    StartTime: new Date(2018, 1, 13, 9, 0),
    EndTime: new Date(2018, 1, 13, 10, 0),
    IsAllDay: false
  }];
  const eventSettings = { dataSource: scheduleData };
  const onAddClick = () => {
    let Data = [{
      Id: 1,
      Subject: 'Conference',
      StartTime: new Date(2018, 1, 12, 9, 0),
      EndTime: new Date(2018, 1, 12, 10, 0),
      IsAllDay: false
    }, {
      Id: 2,
      Subject: 'Meeting',
      StartTime: new Date(2018, 1, 15, 10, 0),
      EndTime: new Date(2018, 1, 15, 11, 30),
      IsAllDay: false
    }];
  };
};
```

```

    scheduleObj.current.addEvent(Data);
    buttonObj.current.element.setAttribute('disabled', 'true');
  }
  return (<div>
    <ButtonComponent id='add' title='Add' ref={buttonObj}
    onClick={onAddClick}>Add</ButtonComponent>
    <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
    selectedDate=
      {new Date(2018, 1, 15)} eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
  </div>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const buttonObj = useRef<ButtonComponent>(null);
  let scheduleData: Object[] = [{
    Id: 3,
    Subject: 'Testing',
    StartTime: new Date(2018, 1, 11, 9, 0),
    EndTime: new Date(2018, 1, 11, 10, 0),
    IsAllDay: false
  }, {
    Id: 4,
    Subject: 'Vacation',
    StartTime: new Date(2018, 1, 13, 9, 0),
    EndTime: new Date(2018, 1, 13, 10, 0),
    IsAllDay: false
  }];
  const eventSettings = { dataSource: scheduleData }
  const onAddClick = (): void => {
    let Data: Object[] = [{
      Id: 1,
      Subject: 'Conference',
      StartTime: new Date(2018, 1, 12, 9, 0),
      EndTime: new Date(2018, 1, 12, 10, 0),
      IsAllDay: false
    }];
  }
}

```

```

    }, {
      Id: 2,
      Subject: 'Meeting',
      StartTime: new Date(2018, 1, 15, 10, 0),
      EndTime: new Date(2018, 1, 15, 11, 30),
      IsAllDay: false
    }];
    scheduleObj.current.addEvent(Data);
    buttonObj.current.element.setAttribute('disabled', 'true');
  }
  return (<div>
    <ButtonComponent id='add' title='Add' ref={buttonObj}
    onClick={onAddClick}>Add</ButtonComponent>
    <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
    selectedDate=
      {new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
  </div>
  );
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-popups/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>

<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>

</html>

```

Inserting events into database at server-side

While adding the normal or recurring events to the Scheduler, **insert** action takes place and the following code example describes how to add a new event into database at server side.

```
`ts
```

```
if (param.action == "insert" || (param.action == "batch" && param.added != null)) // this block of code
will execute while inserting the appointments
```

```
{
```

```
var value = (param.action == "insert") ? param.value : param.added[0];
```

```
int intMax = db.ScheduleEventDatas.ToList().Count > 0 ? db.ScheduleEventDatas.ToList().Max(p => p.Id) :
1;
```

```
DateTime startTime = Convert.ToDateTime(value.StartTime);
```

```
DateTime endTime = Convert.ToDateTime(value.EndTime);
```

```
ScheduleEventData appointment = new ScheduleEventData()
```

```
{
```

```
Id = intMax + 1,
```

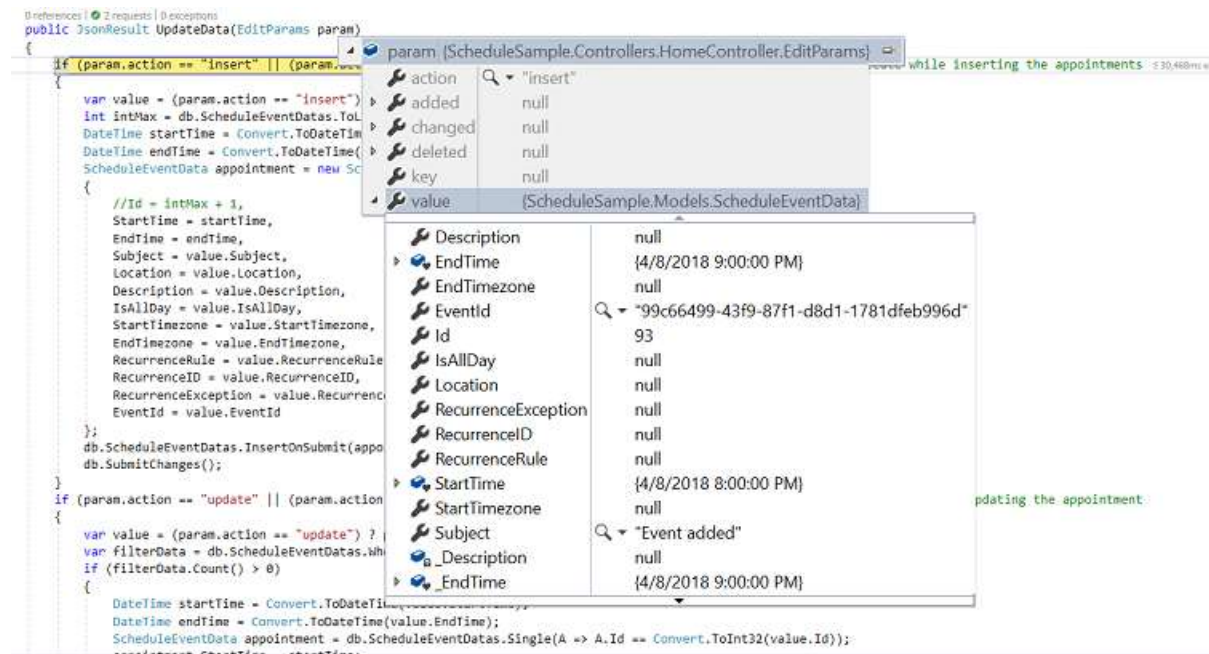
```
StartTime = startTime.ToLocalTime(),
```

```
EndTime = endTime.ToLocalTime(),
```

```

Subject = value.Subject,
IsAllDay = value.IsAllDay,
StartTimezone = value.StartTimezone,
EndTimezone = value.EndTimezone,
RecurrenceRule = value.RecurrenceRule,
RecurrenceID = value.RecurrenceID,
RecurrenceException = value.RecurrenceException
};
db.ScheduleEventDatas.InsertOnSubmit(appointment);
db.SubmitChanges();
}

```



Restricting add action based on specific criteria

In the following example, the specific fields of Scheduler editor window such as Subject and Location are made to undergo validation such that if it is left as blank, then the default **Required** validation message will be displayed, while clicking on a save button.

Additionally, the regex condition has been added to the Location field, so that if any special characters are typed into it, then the custom validation message will be displayed.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
'@syncfusion/ej2-react-schedule';

```



```
import { scheduleData } from './datasource';
const App = () => {
  const fields = {
    subject: { name: 'Subject', validation: { required: true } },
    location: {
      name: 'Location', validation: {
        required: true,
        regex: ["^[a-zA-Z0-9- ]*$", 'Special character(s) not allowed
in this field']
      }
    }
  };
  const eventSettings = { dataSource: scheduleData, fields: fields }
  return (<ScheduleComponent height='550px' width='100%' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const fields = {
    subject: { name: 'Subject', validation: { required: true } },
    location: {
      name: 'Location', validation: {
        required: true,
        regex: ["^[a-zA-Z0-9- ]*$", 'Special character(s) not allowed in this
field']
      }
    }
  };
  const eventSettings: EventSettingsModel = { dataSource: scheduleData,
fields: fields }
  return (<ScheduleComponent height='550px' width='100%' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
      background: green;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

You can also dynamically prevent the creation of appointments on Scheduler. For example, say if you want to decline the creation of appointments on weekend days, you can check for its appropriate condition within the `actionBegin` event.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    const onActionBegin = (args) => {
        let weekEnds = [0, 6];
        if (args.requestType === 'eventCreate' &&
            weekEnds.indexOf((args.data[0].StartTime).getDay()) >= 0) {
            args.cancel = true;
        }
    }
    return (<ScheduleComponent height='550px' width='100%' selectedDate={new
    Date(2018, 1, 15)} eventSettings={eventSettings}
    actionBegin={onActionBegin.bind(this)}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, ActionEventArgs,
    Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData }
    const onActionBegin = (args: ActionEventArgs) => {
        let weekEnds: number[] = [0, 6];
        if (args.requestType === 'eventCreate' &&
            weekEnds.indexOf((args.data[0].StartTime).getDay()) >= 0) {
            args.cancel = true;
        }
    }
    return (
        <ScheduleComponent height='550px' width='100%' selectedDate={new
        Date(2018, 1, 15)} eventSettings={eventSettings}
        actionBegin={onActionBegin.bind(this)}>
            <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
    )
}
```

```

    )
  };
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {

```

```

        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Edit

The same way the appointments such as normal, all-day, spanned or recurring events are created, it can be easily edited using any of the following ways.

- [Update using editor window](#)
- [Update using saveEvent method](#)

Update using editor window

You can open the default editor window filled with appointment details by double clicking on the required events. It gets pre-filled with event options such as Subject, Location, Start and End time, All-day, timezone, description and other recurrence options, from which you can edit the desired field values and, then enter the **Save** button to update it.

You can also single click on appointments, which opens the quick info popup with edit and delete options. Clicking on the **edit** option will open the default editor filled with event details and **delete** option will prompt for delete confirmation.

Updating using saveEvent method

The appointments can be edited and updated manually using the **saveEvent** method. The following code examples shows how to edit the normal and recurring events.

Normal event - Here, an event with ID **3** is edited and its subject is changed with a new text. When the modified data object is passed onto the **saveEvent** method, the changes gets reflected onto the original event. The **Id** field is mandatory in this edit process, where the modified event object should hold the valid **Id** value that exists in the Scheduler data source.

INDEX.JSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DataManager, Query } from '@syncfusion/ej2-data';
const App = () => {
    const scheduleObj = useRef(null);
    const buttonObj = useRef(null);
    const scheduleData = [{
        Id: 3,
        Subject: 'Testing',
        StartTime: new Date(2018, 1, 11, 9, 0),

```

```

        EndTime: new Date(2018, 1, 11, 10, 0),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
    }, {
        Id: 4,
        Subject: 'Vacation',
        StartTime: new Date(2018, 1, 13, 9, 0),
        EndTime: new Date(2018, 1, 13, 10, 0),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=3'
    }
    ];
    const eventSettings = { dataSource: scheduleData }
    const onEditClick = () => {
        let data = new
DataManager(scheduleObj.current.getCurrentViewEvents()).executeLocal(new
Query().where('RecurrenceID', 'equal', 3));
        data[0].Subject = 'Edited';
        scheduleObj.current.saveEvent(data[0], 'EditOccurrence');
        buttonObj.current.element.setAttribute('disabled', 'true');
    }
    return (
        <div>
            <ButtonComponent id='edit' ref={buttonObj} title='Edit'
onClick={onEditClick.bind(this)}>Edit</ButtonComponent>
            <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate=
                {new Date(2018, 1, 15)} eventSettings={eventSettings}>
                <ViewsDirective>
                    <ViewDirective option='Day' />
                    <ViewDirective option='Week' />
                    <ViewDirective option='WorkWeek' />
                    <ViewDirective option='Month' />
                </ViewsDirective>
                <Inject services={[Day, Week, WorkWeek, Month]} />
            </ScheduleComponent>
        </div>
    )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
    ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DataManager, Query } from '@syncfusion/ej2-data';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const buttonObj = useRef<ButtonComponent>(null);
    const scheduleData: Object[] = [{

```

```

    Id: 3,
    Subject: 'Testing',
    StartTime: new Date(2018, 1, 11, 9, 0),
    EndTime: new Date(2018, 1, 11, 10, 0),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
  }, {
    Id: 4,
    Subject: 'Vacation',
    StartTime: new Date(2018, 1, 13, 9, 0),
    EndTime: new Date(2018, 1, 13, 10, 0),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=3'
  }
];
const eventSettings = { dataSource: scheduleData }
const onEditClick = (): void => {
  let data: Object = new
DataManager(scheduleObj.current.getCurrentViewEvents()).executeLocal(new
Query().where('RecurrenceID', 'equal', 3));
  data[0].Subject = 'Edited';
  scheduleObj.current.saveEvent(data[0], 'EditOccurrence');
  buttonObj.current.element.setAttribute('disabled', 'true');
}
return (
  <div>
    <ButtonComponent id='edit' ref={buttonObj} title='Edit'
onClick={onEditClick.bind(this)}>Edit</ButtonComponent>
    <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate=
  {new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>
</div>
)
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
  />
    <meta name="description" content="Essential JS 2 for React
Components" />
    <meta name="author" content="Syncfusion" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>

<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>

</html>

```

Recurring event - The following code example shows how to edit a single occurrence of a recurring event. In this case, the modified data should hold an additional field namely `RecurrenceID` mapping to its parent recurring event's Id value. Also, this modified occurrence will be considered as a new event in the Scheduler dataSource, where it is linked with its parent event through the `RecurrenceID` field value. The `saveEvent` method takes 2 arguments, first one accepting the modified event data object and second argument accepting either of the 2 text values - `EditOccurrence` or `EditSeries`.

When the second argument is passed as `EditOccurrence`, which means that the passed event data is a single modified occurrence - whereas if the second argument is passed as `EditSeries`, it means that the modified data needs to be edited as a whole series and therefore no new event object will be maintained in the Scheduler dataSource.

In case of modifying the single occurrence, it is also necessary to update the `RecurrenceException` field of parent event altogether with the occurrence editing. To know more about how to set `RecurrenceException` values, refer the [recurring events](#) topic.

INDEX.JSX

```
import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DataManager, Query } from '@syncfusion/ej2-data';
const App = () => {
  const scheduleObj = useRef(null);
  const buttonObj = useRef(null);
  const scheduleData = [{
    Id: 3,
    Subject: 'Testing',
    StartTime: new Date(2018, 1, 11, 9, 0),
    EndTime: new Date(2018, 1, 11, 10, 0),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
  }, {
    Id: 4,
    Subject: 'Vacation',
    StartTime: new Date(2018, 1, 13, 9, 0),
    EndTime: new Date(2018, 1, 13, 10, 0),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=3'
  }
  ];
  const eventSettings = { dataSource: scheduleData }
  const onEditClick = () => {
    let data = new
DataManager(scheduleObj.current.getCurrentViewEvents()).executeLocal(new
Query().where('RecurrenceID', 'equal', 3));
    data[0].Subject = 'Edited';
    scheduleObj.current.saveEvent(data[0], 'EditOccurrence');
    buttonObj.current.element.setAttribute('disabled', 'true');
  }
  return (
    <div>
      <ButtonComponent id='edit' ref={buttonObj} title='Edit'
onClick={onEditClick.bind(this)}>Edit</ButtonComponent>
      <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate=
{new Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
          <ViewDirective option='Day' />
          <ViewDirective option='Week' />
          <ViewDirective option='WorkWeek' />
          <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
      </ScheduleComponent>
    </div>
  )
}
```

```
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DataManager, Query } from '@syncfusion/ej2-data';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const buttonObj = useRef<ButtonComponent>(null);
  const scheduleData: Object[] = [{
    Id: 3,
    Subject: 'Testing',
    StartTime: new Date(2018, 1, 11, 9, 0),
    EndTime: new Date(2018, 1, 11, 10, 0),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
  }, {
    Id: 4,
    Subject: 'Vacation',
    StartTime: new Date(2018, 1, 13, 9, 0),
    EndTime: new Date(2018, 1, 13, 10, 0),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=3'
  }];
  const eventSettings = { dataSource: scheduleData };
  const onEditClick = (): void => {
    let data: Object = new
    DataManager(scheduleObj.current.getCurrentViewEvents()).executeLocal(new
    Query().where('RecurrenceID', 'equal', 3));
    data[0].Subject = 'Edited';
    scheduleObj.current.saveEvent(data[0], 'EditOccurrence');
    buttonObj.current.element.setAttribute('disabled', 'true');
  }
  return (
    <div>
      <ButtonComponent id='edit' ref={buttonObj} title='Edit'
      onClick={onEditClick.bind(this)}>Edit</ButtonComponent>
      <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
      selectedDate=
        {new Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
          <ViewDirective option='Day' />
          <ViewDirective option='Week' />
          <ViewDirective option='WorkWeek' />
          <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
      </ScheduleComponent>
    </div>
  );
}
```

```

    </ScheduleComponent>
  </div>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-schedule/styles/material.css" rel="stylesheet" />
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />
    <script src="https://cdn.jsdelivr.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></script>
    <script src="systemjs.config.js"></script>
    <style>
      #loader {
        color: #008c8f;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
      }
    </style>
  </head>

  <body>
    <div id='schedule'>

```

```

        <div id='loader'>Loading....</div>
      </div>
    </body>

  </html>

```

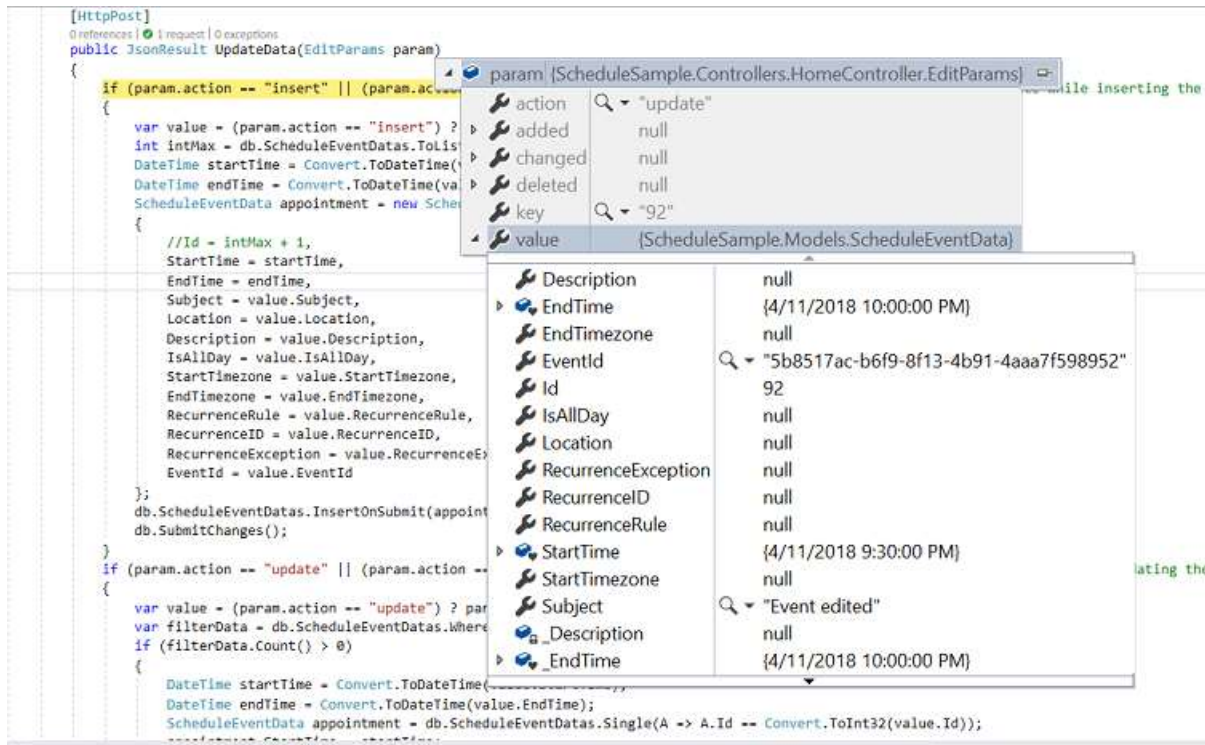
Updating events in database at server-side

While editing the normal events in the Scheduler, **update** action takes place and the following code example describes how to update event into database at server side.

```

`ts
if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of
code will execute while updating the appointment
{
var value = (param.action == "update") ? param.value : param.changed[0];
var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
if (filterData.Count() > 0)
{
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
Convert.ToInt32(value.Id));
appointment.StartTime = startTime.ToLocalTime();
appointment.EndTime = endTime.ToLocalTime();
appointment.StartTimezone = value.StartTimezone;
appointment.EndTimezone = value.EndTimezone;
appointment.Subject = value.Subject;
appointment.IsAllDay = value.IsAllDay;
appointment.RecurrenceRule = value.RecurrenceRule;
appointment.RecurrenceID = value.RecurrenceID;
appointment.RecurrenceException = value.RecurrenceException;
}
db.SubmitChanges();
}
`

```



How to edit a single occurrence or entire series and update it in database at server-side

The recurring appointments can be edited in either of the following two ways.

- Single occurrence
- Entire series

Editing single occurrence - When you double click on a recurring event, a popup prompts you to choose either to edit the single event or entire series. From this, if you choose to select **EDIT EVENT** option, a single occurrence of the recurring appointment alone will be edited. The following process takes place while editing a single occurrence,

- A new event will be created from the parent event data and added to the Scheduler dataSource, with all its default field values overwritten with the newly modified data and additionally, the **recurrenceID** field will be added to it, that holds the **id** value of the parent recurring event. Also, a new **Id** will be generated for this event in the dataSource.
- The parent recurring event needs to be updated with appropriate **recurrenceException** field to hold the edited occurrence appointment's date collection.

Therefore, when a single occurrence is edited from a recurring event, the batch action takes place by allowing both the **Add** and **Edit** action requests to take place together.

In case, if you edit an existing edited occurrence of a recurring event, only those edited occurrence which present in the database as an individual event object will get updated. In this case, **update** action alone takes place on the edited occurrence object on the database.

`ts

```
if (param.action == "insert" || (param.action == "batch" && param.added != null)) // this block of code
will execute while inserting the appointments
{
var value = (param.action == "insert") ? param.value : param.added[0];
int intMax = db.ScheduleEventDatas.ToList().Count > 0 ? db.ScheduleEventDatas.ToList().Max(p => p.Id) :
1;
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = new ScheduleEventData()
{
Id = intMax + 1,
StartTime = startTime.ToLocalTime(),
EndTime = endTime.ToLocalTime(),
Subject = value.Subject,
IsAllDay = value.IsAllDay,
StartTimezone = value.StartTimezone,
EndTimezone = value.EndTimezone,
RecurrenceRule = value.RecurrenceRule,
RecurrenceID = value.RecurrenceID,
RecurrenceException = value.RecurrenceException
};
db.ScheduleEventDatas.InsertOnSubmit(appointment);
db.SubmitChanges();
}

if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of
code will execute while updating the appointment
{
var value = (param.action == "update") ? param.value : param.changed[0];
var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
if (filterData.Count() > 0)
{
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
```

```

ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
Convert.ToInt32(value.Id));
appointment.StartTime = startTime.ToLocalTime();
appointment.EndTime = endTime.ToLocalTime();
appointment.StartTimezone = value.StartTimezone;
appointment.EndTimezone = value.EndTimezone;
appointment.Subject = value.Subject;
appointment.IsAllDay = value.IsAllDay;
appointment.RecurrenceRule = value.RecurrenceRule;
appointment.RecurrenceID = value.RecurrenceID;
appointment.RecurrenceException = value.RecurrenceException;
}
db.SubmitChanges();
}
,

```

Editing entire series - When you select an option **EDIT SERIES** from the popup that opens on double clicking the recurring event, the whole recurring series will be updated with the newly provided value. When this option is chosen explicitly, if a parent event holds any edited occurrences - then all its child occurrences will be removed from the dataSource and simply the single parent data will be updated.

This action of editing entire series also leads to the batch process, as both the **Delete** and **Edit** action takes place together.

```
`ts
```

```

if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of
code will execute while updating the appointment
{
var value = (param.action == "update") ? param.value : param.changed[0];
var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
if (filterData.Count() > 0)
{
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
Convert.ToInt32(value.Id));
appointment.StartTime = startTime.ToLocalTime();
appointment.EndTime = endTime.ToLocalTime();

```

```

appointment.StartTimezone = value.StartTimezone;
appointment.EndTimezone = value.EndTimezone;
appointment.Subject = value.Subject;
appointment.IsAllDay = value.IsAllDay;
appointment.RecurrenceRule = value.RecurrenceRule;
appointment.RecurrenceID = value.RecurrenceID;
appointment.RecurrenceException = value.RecurrenceException;
}
db.SubmitChanges();
}

if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of
code will execute while removing the appointment
{
if (param.action == "remove")
{
int key = Convert.ToInt32(param.key);
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
else
{
foreach (var apps in param.deleted)
{
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
}
db.SubmitChanges();
}

```

How to edit from the current and following events of a series

The recurring appointments can be edited from current and following events when enable the property editFollowingEvents.

Editing Following Events - When you double click on a recurring event, a popup prompts you to choose either to edit the single event or Edit Following Events or entire series. From this, if you choose to select **EDIT FOLLOWING EVENTS** option, a current and following events of the recurring appointment will be edited. The following process takes place while editing a following events,

- A new event will be created from the parent event data and added to the Scheduler dataSource, with all its default field values overwritten with the newly modified data and additionally, the **followingID** field will be added to it, that holds the **id** value of the immediate parent recurring event. Also, a new **Id** will be generated for this event in the dataSource.
- The parent recurring event needs to be updated with appropriate **recurrenceRule** field to hold the modified occurrence appointment's end date.

Therefore, when a following events are edited from a recurring event, the batch action takes place by allowing the **Add**, **Edit** and **Delete** action requests to take place together.

```
`ts
```

```
if (param.action == "insert" || (param.action == "batch" && param.added != null)) // this block of code
will execute while inserting the appointments
{
var value = (param.action == "insert") ? param.value : param.added[0];
int intMax = db.ScheduleEventDatas.ToList().Count > 0 ? db.ScheduleEventDatas.ToList().Max(p => p.Id) :
1;
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = new ScheduleEventData()
{
Id = intMax + 1,
StartTime = startTime.ToLocalTime(),
EndTime = endTime.ToLocalTime(),
Subject = value.Subject,
IsAllDay = value.IsAllDay,
StartTimezone = value.StartTimezone,
EndTimezone = value.EndTimezone,
RecurrenceRule = value.RecurrenceRule,
RecurrenceID = value.RecurrenceID,
FollowingID = value.FollowingID,
RecurrenceException = value.RecurrenceException
};
```

```
db.ScheduleEventDatas.InsertOnSubmit(appointment);
db.SubmitChanges();
}

if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of
code will execute while updating the appointment
{
    var value = (param.action == "update") ? param.value : param.changed[0];
    var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
    if (filterData.Count() > 0)
    {
        DateTime startTime = Convert.ToDateTime(value.StartTime);
        DateTime endTime = Convert.ToDateTime(value.EndTime);
        ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
        Convert.ToInt32(value.Id));
        appointment.StartTime = startTime.ToLocalTime();
        appointment.EndTime = endTime.ToLocalTime();
        appointment.StartTimezone = value.StartTimezone;
        appointment.EndTimezone = value.EndTimezone;
        appointment.Subject = value.Subject;
        appointment.IsAllDay = value.IsAllDay;
        appointment.RecurrenceRule = value.RecurrenceRule;
        appointment.RecurrenceID = value.RecurrenceID;
        appointment.FollowingID = value.FollowingID;
        appointment.RecurrenceException = value.RecurrenceException;
    }
    db.SubmitChanges();
}

if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of
code will execute while removing the appointment
{
    if (param.action == "remove")
    {
        int key = Convert.ToInt32(param.key);
        ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
```

```

if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
else
{
foreach (var apps in param.deleted)
{
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
}
db.SubmitChanges();
}
`

```

To know more about handling recurrence exceptions, refer the [Adding exceptions](#) topic.

Restricting edit action based on specific criteria

You can also dynamically prevent the editing of appointments on Scheduler. For example, say if you want to decline the updating of appointments on non-working hours, you can check for its appropriate condition within the `actionBegin` event.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onActionBegin = (args) => {
    if (args.requestType === 'eventChange') {
      let weekEnds = [0, 6];
      let weekDay = weekEnds.indexOf((args.data.StartTime).getDay()) >= 0;
      let workHours = (args.data.StartTime.getHours <
      parseInt(scheduleObj.current.workHours.start)) ||
      (args.data.StartTime.getHours > parseInt(scheduleObj.current.workHours.end));
      if (weekDay || workHours) {
        args.cancel = true;
      }
    }
  }
  return (<ScheduleComponent height='550px' width='100%' selectedDate={new
  Date(2018, 1, 15)} ref={scheduleObj} eventSettings={eventSettings}
  actionBegin={onActionBegin.bind(this)}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />

```

```

    </ScheduleComponent>);
  }
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, ActionEventArgs,
  Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  const onActionBegin = (args: ActionEventArgs): void => {
    if (args.requestType === 'eventChange') {
      let weekEnds: number[] = [0, 6];
      let weekDay: boolean = weekEnds.indexOf((args.data as any).StartTime.getDay()) >= 0;
      let workHours: boolean = ((args.data as any).StartTime.getHours <
        parseInt(scheduleObj.current.workHours.start)) || ((args.data as any).StartTime.getHours >
        parseInt(scheduleObj.current.workHours.end));
      if (weekDay || workHours) {
        args.cancel = true;
      }
    }
  }
  return (
    <ScheduleComponent height='550px' width='100%' selectedDate={new Date(2018, 1, 15)} ref={scheduleObj} eventSettings={eventSettings}
    actionBegin={onActionBegin.bind(this)}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Delete

The appointments can be deleted in either of the following ways,

- Selecting an appointment and clicking the delete icon from the quick popup that opens.
- Selecting an appointment and pressing **Delete** key.

- Selecting multiple appointments by tap holding an event and then continuously single clicking on other consecutive events and then clicking the **Delete** key.
- Double clicking on an event which opens the default event editor pre-filled with event details, and then choosing **Delete** button in it.

While performing all these above mentioned actions, a pop-up with a delete confirmation message will be displayed prompting either to proceed with deleting an appointment.

Deletion using editor window

When you double click an event, the default editor window will be opened which includes a **Delete** button at the bottom left position to allow you to delete that particular appointment. When deleting an appointment through this editor window, the delete alert confirmation will not be asked and the event will be deleted immediately.

Deletion using deleteEvent method

The appointments can be removed manually using the **deleteEvent** method. The following code examples shows how to edit the normal and recurring events.

Normal event - You can delete the normal appointments of Scheduler by simply passing its **Id** value or the entire event object collection to the **deleteEvent** method.

INDEX.JSX

```
import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef(null);
  const buttonObj = useRef(null);
  const scheduleData = [{
    Id: 3,
    Subject: 'Testing',
    StartTime: new Date(2018, 1, 11, 9, 0),
    EndTime: new Date(2018, 1, 11, 10, 0),
    IsAllDay: false
  }, {
    Id: 4,
    Subject: 'Vacation',
    StartTime: new Date(2018, 1, 13, 9, 0),
    EndTime: new Date(2018, 1, 13, 10, 0),
    IsAllDay: false
  }
  ];
  const eventSettings = { dataSource: scheduleData };
  const onDeleteClick = () => {
    scheduleObj.current.deleteEvent(4);
    buttonObj.current.element.setAttribute('disabled', 'true');
  }
  return (
    <div>
      <ButtonComponent id='delete' ref={buttonObj} title='Delete'
onOnClick={onDeleteClick}>Delete</ButtonComponent>
```

```

    <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate=
    {new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
</div>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
    ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const buttonObj = useRef<ButtonComponent>(null);
    const scheduleData: Object[] = [{
        Id: 3,
        Subject: 'Testing',
        StartTime: new Date(2018, 1, 11, 9, 0),
        EndTime: new Date(2018, 1, 11, 10, 0),
        IsAllDay: false
    }, {
        Id: 4,
        Subject: 'Vacation',
        StartTime: new Date(2018, 1, 13, 9, 0),
        EndTime: new Date(2018, 1, 13, 10, 0),
        IsAllDay: false
    }
    ];
    const eventSettings = { dataSource: scheduleData };
    const onDeleteClick = (): void => {
        scheduleObj.current.deleteEvent(4);
        buttonObj.current.element.setAttribute('disabled', 'true');
    }
    return (
        <div>
            <ButtonComponent id='delete' ref={buttonObj} title='Delete'
onClick={onDeleteClick}>Delete</ButtonComponent>
            <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate=
                {new Date(2018, 1, 15)} eventSettings={eventSettings}>
                <ViewsDirective>

```

```

        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
  </div>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-schedule/styles/material.css" rel="stylesheet" />
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />
    <script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></script>
    <script src="systemjs.config.js"></script>
    <style>
      #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
```



```

    }
  </style>
</head>

<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>

</html>

```

Recurring Event - The recurring events can be removed as an entire series or simply removing single occurrence by using the deleteEvent method which takes in either the `DeleteSeries` or `DeleteOccurrence` parameters. The following code example shows how to delete entire series.

INDEX.JSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef(null);
  const buttonObj = useRef(null);
  const scheduleData = [{
    Id: 3,
    Subject: 'Testing',
    StartTime: new Date(2018, 1, 11, 9, 0),
    EndTime: new Date(2018, 1, 11, 10, 0),
    IsAllDay: false
  }, {
    Id: 4,
    Subject: 'Vacation',
    StartTime: new Date(2018, 1, 13, 9, 0),
    EndTime: new Date(2018, 1, 13, 10, 0),
    IsAllDay: false
  }];
  const eventSettings = { dataSource: scheduleData }
  const onDeleteClick = () => {
    scheduleObj.current.deleteEvent(4);
    buttonObj.current.element.setAttribute('disabled', 'true');
  }
  return (
    <div>
      <ButtonComponent id='delete' ref={buttonObj} title='Delete'
onClick={onDeleteClick}>Delete</ButtonComponent>
      <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate=
      {new Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
          <ViewDirective option='Day' />
          <ViewDirective option='Week' />
          <ViewDirective option='WorkWeek' />

```

```

        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
  </div>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const buttonObj = useRef<ButtonComponent>(null);
  const scheduleData: Object[] = [{
    Id: 3,
    Subject: 'Testing',
    StartTime: new Date(2018, 1, 11, 9, 0),
    EndTime: new Date(2018, 1, 11, 10, 0),
    IsAllDay: false
  }, {
    Id: 4,
    Subject: 'Vacation',
    StartTime: new Date(2018, 1, 13, 9, 0),
    EndTime: new Date(2018, 1, 13, 10, 0),
    IsAllDay: false
  }];
  const eventSettings = { dataSource: scheduleData };
  const onDeleteClick = (): void => {
    scheduleObj.current.deleteEvent(4);
    buttonObj.current.element.setAttribute('disabled', 'true');
  }
  return (
    <div>
      <ButtonComponent id='delete' ref={buttonObj} title='Delete'
        onClick={onDeleteClick}>Delete</ButtonComponent>
      <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
        selectedDate=
          {new Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
          <ViewDirective option='Day' />
          <ViewDirective option='Week' />
          <ViewDirective option='WorkWeek' />
          <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
      </ScheduleComponent>
    </div>
  );
};

```

```

    </div>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-schedule/styles/material.css" rel="stylesheet" />
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />
    <script src="https://cdn.jsdelivr.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></script>
    <script src="systemjs.config.js"></script>
    <style>
      #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
      }
    </style>
  </head>

  <body>
    <div id='schedule'>
      <div id='loader'>Loading....</div>
    </div>
  </body>
</html>

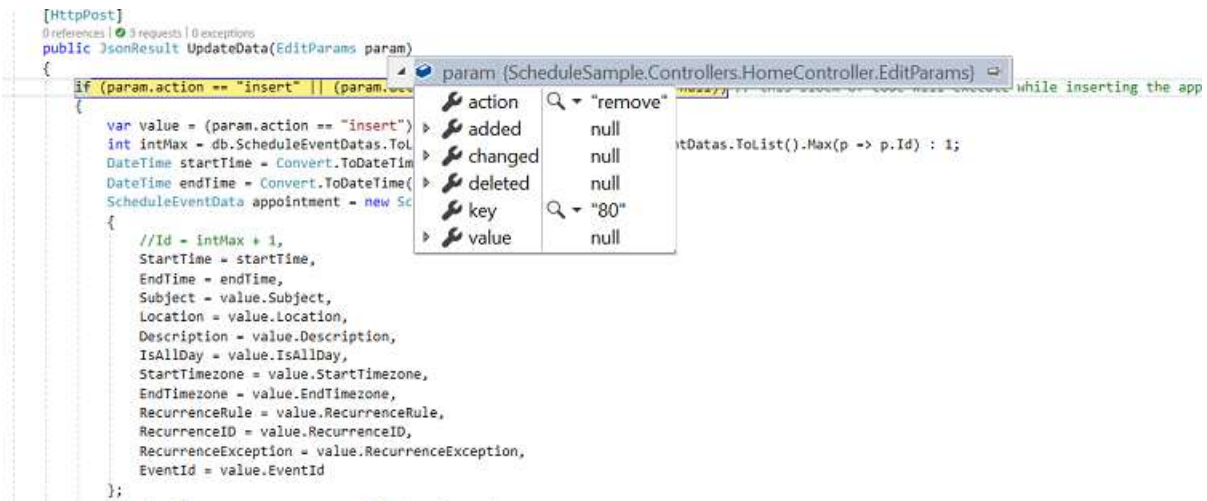
```

```
        </div>
    </body>
</html>
```

Removing events from database at server-side

While deleting the event from the Scheduler, **remove** action takes place and the following code example describes how to delete event from database at server side.

```
`ts
if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of
code will execute while removing the appointment
{
    if (param.action == "remove")
    {
        int key = Convert.ToInt32(param.key);
        ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
        if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
    }
    else
    {
        foreach (var apps in param.deleted)
        {
            ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
            if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
        }
    }
    db.SubmitChanges();
}
```



How to delete a single occurrence or entire series from Scheduler and update it in database at server-side

The recurring events can be deleted in either of the following two ways.

- Single occurrence
- Entire series

Single occurrence - When you attempt to delete the recurring events, a popup prompts you to choose either to delete the single event or entire series. From this, if you choose to select **DELETE EVENT** option, a single occurrence of the recurring appointment alone will be removed. The following process takes place while removing a single occurrence,

- The selected occurrence will be deleted from the Scheduler user interface.
- In code, the parent recurring event object will be updated with appropriate `recurrenceException` field, to hold the deleted occurrence appointment's date collection.

Therefore, when a single occurrence is deleted from a recurring event, the **update** action takes place on the parent recurring event as shown in the following code example.

In case, if you delete an existing edited occurrence of a recurring event, only those edited occurrence which present in the database as an individual event object will get removed. In this case, **delete** action takes place instead of **update** action and the parent recurring event object remains same with no changes.

`ts

if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of code will execute while updating the appointment

{

```
var value = (param.action == "update") ? param.value : param.changed[0];
```

```
var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
```

```
if (filterData.Count() > 0)
```

{

```

DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
Convert.ToInt32(value.Id));
appointment.StartTime = startTime.ToLocalTime();
appointment.EndTime = endTime.ToLocalTime();
appointment.StartTimezone = value.StartTimezone;
appointment.EndTimezone = value.EndTimezone;
appointment.Subject = value.Subject;
appointment.IsAllDay = value.IsAllDay;
appointment.RecurrenceRule = value.RecurrenceRule;
appointment.RecurrenceID = value.RecurrenceID;
appointment.RecurrenceException = value.RecurrenceException;
}
db.SubmitChanges();
}
`

```

Entire series - When you select an option **DELETE SERIES** from the popup, the whole recurring series will be deleted. When this option is chosen explicitly, if a parent event holds any edited occurrences - then all its child occurrences which are maintained as separate event objects will also be removed from the dataSource. This action of deleting entire series leads to **remove** action and removes one or more event objects at the same time.

```
`ts
```

```
if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of
code will execute while removing the appointment
```

```

{
if (param.action == "remove")
{
int key = Convert.ToInt32(param.key);
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
else
{
foreach (var apps in param.deleted)

```

```

{
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
}
db.SubmitChanges();
}
,

```

How to delete only the current and following events of a series

The recurring events can be deleted from current and following events only when enable `editFollowingEvents` property.

Delete Following Events - When you attempt to delete the recurring events, a popup prompts you to choose either to delete the single event or Following Events or entire series. From this, if you choose to select **FOLLOWING EVENT** option, a current and following events of the recurring appointment alone will be removed. The following process takes place while removing a single occurrence,

- The selected occurrence and the following events in same series will be deleted from the Scheduler user interface.
- In code, the parent recurring event object will be updated with appropriate `recurrenceRule` field, to update the end date of the recurring events.

Therefore, when following events are deleted from a recurring event, the `remove` and `update` action takes place on the immediate parent recurring event as shown in the following code example.

```

`ts
if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of
code will execute while updating the appointment
{
var value = (param.action == "update") ? param.value : param.changed[0];
var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
if (filterData.Count() > 0)
{
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
Convert.ToInt32(value.Id));
appointment.StartTime = startTime.ToLocalTime();
appointment.EndTime = endTime.ToLocalTime();
appointment.StartTimezone = value.StartTimezone;

```

```

appointment.EndTimezone = value.EndTimezone;
appointment.Subject = value.Subject;
appointment.IsAllDay = value.IsAllDay;
appointment.RecurrenceRule = value.RecurrenceRule;
appointment.RecurrenceID = value.RecurrenceID;
appointment.FollowingID = value.FollowingID;
appointment.RecurrenceException = value.RecurrenceException;
}
db.SubmitChanges();
}

if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of
code will execute while removing the appointment
{
if (param.action == "remove")
{
int key = Convert.ToInt32(param.key);
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
else
{
foreach (var apps in param.deleted)
{
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
}
db.SubmitChanges();
}

```

Drag and drop

When you drag and drop a normal event on the Scheduler, the event editing action takes place. When a recurring event is drag and dropped on a desired time range, the batch action explained in [Editing a single occurrence](#) process will take place - thus allowing both the [Add](#) and [Edit](#) action to take place together.

By default, when you drag a recurring instance, only the occurrence of the event gets edited and not a whole series.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, TimelineViews, TimelineMonth, Month,
Agenda, DragAndDrop, ViewsDirective, ViewDirective, Inject } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData }
  return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineMonth' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews, TimelineMonth, Month,
Agenda, DragAndDrop]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, TimelineViews, TimelineMonth, Month, Agenda,
  DragAndDrop, ViewsDirective, ViewDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='TimelineDay' />
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='TimelineMonth' />
        <ViewDirective option='Month' />
        <ViewDirective option='Agenda' />
      </ViewsDirective>
      <Inject services={[Day, Week, TimelineViews, TimelineMonth, Month,
Agenda, DragAndDrop]} />
    </ScheduleComponent>
  );
}
```

```

    )
  };
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {

```

```

        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Resize

When you resize a normal event on the Scheduler, the event editing action takes place. When a recurring event is resized to a new desired time, the batch action explained in [Editing a single occurrence](#) process will take place - thus allowing both the [Add](#) and [Edit](#) action to take place together.

By default, when you resize a recurring instance, only the occurrence of the event gets edited and not a whole series.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, TimelineViews, TimelineMonth, Month,
Agenda, Resize, ViewsDirective, ViewDirective, Inject } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='TimelineWeek' />
            <ViewDirective option='Week' />
            <ViewDirective option='TimelineMonth' />
            <ViewDirective option='Month' />
            <ViewDirective option='Agenda' />
        </ViewsDirective>
        <Inject services={[Day, Week, TimelineViews, TimelineMonth, Month,
Agenda, Resize]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, TimelineViews, TimelineMonth, Month, Agenda,
    Resize, ViewsDirective, ViewDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';

```

```
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)}
    eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='Week' />
        <ViewDirective option='TimelineMonth' />
        <ViewDirective option='Month' />
        <ViewDirective option='Agenda' />
      </ViewsDirective>
      <Inject services={[Day, Week, TimelineViews, TimelineMonth, Month,
Agenda, Resize]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
```

```

<style>
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
  .e-past-app {
    background-color: chocolate !important;
  }
  .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
  .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
  .custom-class.e-schedule .e-month-view .e-appointment {
    background: green;
  }
</style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Virtual scrolling in React Schedule component

To achieve better performance in the Scheduler when loading a large number of resources and events, we have added virtual scrolling support to load a large set of resources and events instantly as you scroll. You can dynamically load large number of resources and events in the Scheduler by setting `true` to the `allowVirtualScrolling` property within the view specific settings. The virtual loading of events is possible in Agenda view, by setting `allowVirtualScrolling` property to `true` within the agenda view specific settings.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective,
ResourcesDirective, ResourceDirective, TimelineMonth, TimelineYear, Resize,
DragAndDrop, Inject } from '@syncfusion/ej2-react-schedule';
const App = () => {
  const generateStaticEvents = (start, resCount, overlapCount) => {
    let data = [];
    let id = 1;
    for (let i = 0; i < resCount; i++) {
      let randomCollection = [];
      let random = 0;
      for (let j = 0; j < overlapCount; j++) {
        random = Math.floor(Math.random() * (30));

```

```

        random = (random === 0) ? 1 : random;
        if (randomCollection.indexOf(random) !== -1 ||
randomCollection.indexOf(random + 2) !== -1 ||
        randomCollection.indexOf(random - 2) !== -1) {
            random += (Math.max.apply(null, randomCollection) + 10);
        }
        for (let k = 1; k <= 2; k++) {
            randomCollection.push(random + k);
        }
        let startDate = new Date(start.getFullYear(), start.getMonth(),
random);
        startDate = new Date(startDate.getTime() + (((random % 10) * 10) *
(1000 * 60)));
        let endDate = new Date(startDate.getTime() + ((1440 + 30) * (1000 *
60)));
        data.push({
            Id: id,
            Subject: 'Event #' + id,
            StartTime: startDate,
            EndTime: endDate,
            IsAllDay: (id % 10) ? false : true,
            ResourceId: i + 1
        });
        id++;
    }
}
return data;
}
const eventSettings = { dataSource: generateStaticEvents(new Date(2018, 4,
1), 300, 12) }
const group = { resources: ['Resources'] };
const generateResourceData = (startId, endId, text) => {
    let data = [];
    let colors = [
        '#ff8787', '#9775fa', '#748ffc', '#3bc9db', '#69db7c',
        '#fdd835', '#748ffc', '#9775fa', '#df5286', '#7fa900',
        '#fec200', '#5978ee', '#00bdae', '#ea80fc'
    ];
    for (let a = startId; a <= endId; a++) {
        let n = Math.floor(Math.random() * colors.length);
        data.push({
            Id: a,
            Text: text + ' ' + a,
            Color: colors[n]
        });
    }
    return data;
}
return (<ScheduleComponent cssClass='virtual-scrolling' width='100%'
height='550px' selectedDate={new Date(2018, 4, 1)} eventSettings={
    eventSettings} group={group}>
    <ResourcesDirective>
        <ResourceDirective field='ResourceId' title='Resource' name='Resources'
allowMultiple={true} dataSource={generateResourceData(1, 300, 'Resource')}
textField='Text' idField='Id' colorField='Color'>
            </ResourceDirective>
        </ResourcesDirective>
    </ScheduleComponent>

```

```

    <ViewsDirective>
      <ViewDirective option='TimelineMonth' allowVirtualScrolling={true}
isSelected={true} />
      <ViewDirective option='TimelineYear' orientation='Vertical'
allowVirtualScrolling={true} />
    </ViewsDirective>
    <Inject services={[TimelineMonth, TimelineYear, Resize, DragAndDrop]} />
  </ScheduleComponent>);
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
  ScheduleComponent, ViewsDirective, ViewDirective, ResourcesDirective,
  ResourceDirective, TimelineMonth, TimelineYear, Resize, DragAndDrop,
  Inject, EventSettingsModel, GroupModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const generateStaticEvents = (start: Date, resCount: number, overlapCount:
number): Object[] => {
    let data: Object[] = [];
    let id: number = 1;
    for (let i: number = 0; i < resCount; i++) {
      let randomCollection: number[] = [];
      let random: number = 0;
      for (let j: number = 0; j < overlapCount; j++) {
        random = Math.floor(Math.random() * (30));
        random = (random === 0) ? 1 : random;
        if (randomCollection.indexOf(random) !== -1 ||
randomCollection.indexOf(random + 2) !== -1 ||
randomCollection.indexOf(random - 2) !== -1) {
          random += (Math.max.apply(null, randomCollection) + 10);
        }
        for (let k: number = 1; k <= 2; k++) {
          randomCollection.push(random + k);
        }
        let startDate: Date = new Date(start.getFullYear(), start.getMonth(),
random);
        startDate = new Date(startDate.getTime() + (((random % 10) * 10) *
(1000 * 60)));
        let endDate: Date = new Date(startDate.getTime() + ((1440 + 30) *
(1000 * 60)));
        data.push({
          Id: id,
          Subject: 'Event #' + id,
          StartTime: startDate,
          EndTime: endDate,
          IsAllDay: (id % 10) ? false : true,
          ResourceId: i + 1
        });
        id++;
      }
    }
  }
}

```

```

    }
    return data;
  }
  const eventSettings: EventSettingsModel = { dataSource:
generateStaticEvents(new Date(2018, 4, 1), 300, 12) };
  const group: GroupModel = { resources: ['Resources'] };
  const generateResourceData = (startId: number, endId: number, text:
string): Object[] => {
    let data: { [key: string]: Object }[] = [];
    let colors: string[] = [
      '#ff8787', '#9775fa', '#748ffc', '#3bc9db', '#69db7c',
      '#fdd835', '#748ffc', '#9775fa', '#df5286', '#7fa900',
      '#fec200', '#5978ee', '#00bdae', '#ea80fc'
    ];
    for (let a: number = startId; a <= endId; a++) {
      let n: number = Math.floor(Math.random() * colors.length);
      data.push({
        Id: a,
        Text: text + ' ' + a,
        Color: colors[n]
      });
    }
    return data;
  }
  return (
    <ScheduleComponent cssClass='virtual-scrolling' width='100%'
      height='550px' selectedDate={new Date(2018, 4, 1)}
      eventSettings={eventSettings}
      group={group} >
      <ResourcesDirective>
        <ResourceDirective field='ResourceId' title='Resource'
name='Resources' allowMultiple={true}
          dataSource={generateResourceData(1, 300, 'Resource')}
          textField='Text' idField='Id' colorField='Color'>
        </ResourceDirective>
      </ResourcesDirective>
      < ViewsDirective >
        <ViewDirective option='TimelineMonth' allowVirtualScrolling={true}
isSelected={true} />
        <ViewDirective option='TimelineYear' orientation='Vertical'
allowVirtualScrolling={true} />
      </ViewsDirective>
      < Inject services={[TimelineMonth, TimelineYear, Resize, DragAndDrop]}
/>
    </ScheduleComponent>
  );
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>

```



```

<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta name="description" content="Essential JS 2 for React Components" />
<meta name="author" content="Syncfusion" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

For now, the virtual loading of resources and events is not supported in **MonthAgenda**, **Year** and **TimelineYear** (Horizontal Orientation) views.

Enabling lazy loading for appointments

The lazy loading feature provides a convenient way to efficiently load resource appointments into the Scheduler using an on-demand approach. With this feature, you can seamlessly load a large volume of appointment data into the Scheduler without experiencing any performance degradation.

By default, the Scheduler fetches all the relevant appointments from the server with in the current date range. However, enabling this feature will trigger query requests to the server for appointment retrieval whenever new resources are rendered due to scroll actions. These queries contain the resource IDs of currently displayed resources along with current date range, which can be passed as a comma-separated string. In the server controller, these resource IDs are parsed to filter the necessary appointments to render in the scheduler.

When you enable this feature, the Scheduler becomes capable of fetching events from remote services only for the current view port alone to optimize the data retrieval. The remaining appointment data is fetched form the server on-demand based on currently rendered view port resources as you scroll's through the scheduler content.

To enable this feature, you have to set the [enableLazyLoading](#) property to **true** within the view specific settings.

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective,
ResourcesDirective, ResourceDirective, TimelineMonth, Inject } from
'@syncfusion/ej2-react-schedule';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
const App = () => {
  const dataManager = new DataManager({
    url:
'https://services.syncfusion.com/react/production/api/VirtualEventData',
    adaptor: new WebApiAdaptor,
    crossDomain: true
  });
  const eventSettings = { dataSource: dataManager };
  const group = { resources: ['Resources'] };
  const generateResourceData = (startId, endId, text) => {
    let data = [];
    let colors = [
      '#ff8787', '#9775fa', '#748ffc', '#3bc9db', '#69db7c',
      '#fdd835', '#748ffc', '#9775fa', '#df5286', '#7fa900',
      '#fec200', '#5978ee', '#00bdae', '#ea80fc'
    ];
    for (let a = startId; a <= endId; a++) {
      let n = Math.floor(Math.random() * colors.length);
      data.push({
        Id: a,
        Text: text + ' ' + a,
        Color: colors[n]
      });
    }
  }
}
```

```

    return data;
  }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2023, 3, 1)} eventSettings={
  eventSettings} group={group} readonly={true}>
    <ResourcesDirective>
      <ResourceDirective field='ResourceId' title='Resource' name='Resources'
dataSource={generateResourceData(1, 1000, 'Resource')} textField='Text'
idField='Id' colorField='Color'>
        </ResourceDirective>
      </ResourcesDirective>
    <ViewsDirective>
      <ViewDirective option='TimelineMonth' enableLazyLoading={true}
isSelected={true} />
    </ViewsDirective>
    <Inject services={[TimelineMonth]} />
  </ScheduleComponent>);
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
  ScheduleComponent, ViewsDirective, ViewDirective, ResourcesDirective,
  ResourceDirective, TimelineMonth, Inject, EventSettingsModel, GroupModel
} from '@syncfusion/ej2-react-schedule';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
const App = () => {
  const dataManager: DataManager = new DataManager({
    url:
'https://services.syncfusion.com/react/production/api/VirtualEventData',
    adaptor: new WebApiAdaptor,
    crossDomain: true
  });
  const eventSettings: EventSettingsModel = { dataSource: dataManager };
  const group: GroupModel = { resources: ['Resources'] };
  const generateResourceData = (startId: number, endId: number, text:
string): Object[] => {
    let data: { [key: string]: Object }[] = [];
    let colors: string[] = [
      '#ff8787', '#9775fa', '#748ffc', '#3bc9db', '#69db7c',
      '#fdd835', '#748ffc', '#9775fa', '#df5286', '#7fa900',
      '#fec200', '#5978ee', '#00bdae', '#ea80fc'
    ];
    for (let a: number = startId; a <= endId; a++) {
      let n: number = Math.floor(Math.random() * colors.length);
      data.push({
        Id: a,
        Text: text + ' ' + a,
        Color: colors[n]
      });
    }
    return data;
  }
}

```

```

    }
    return (
      <ScheduleComponent width='100%'
        height='550px' selectedDate={new Date(2023, 3, 1)}
        eventSettings={eventSettings}
        group={group} readonly={true}>
        <ResourcesDirective>
          <ResourceDirective field='ResourceId' title='Resource'
name='Resources'
            dataSource={generateResourceData(1, 1000, 'Resource')}
            textField='Text' idField='Id' colorField='Color'>
          </ResourceDirective>
        </ResourcesDirective>
        < ViewsDirective >
          <ViewDirective option='TimelineMonth' enableLazyLoading={true}
isSelected={true} />
        </ViewsDirective>
        < Inject services={[TimelineMonth]} />
      </ScheduleComponent>
    );
  }
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>

```

```

<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Here's the server-side controller code that retrieves appointment data based on the resource IDs provided as query parameters:

```

`c#
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.OData.Query;
namespace LazyLoadingServices.Controllers
{
    public class VirtualEventDataController : Controller
    {
        private readonly EventsContext dbContext;

        [HttpGet]
        [EnableQuery]
        [Route("api/VirtualEventData")]
        public IActionResult GetData([FromQuery] Params param)
        {
            IQueryable<EventData> query = dbContext.Events;

```

```
// Filter the appointment data based on the ResourceId query params.
if (!string.IsNullOrEmpty(param.ResourceId))
{
    string[] resourceId = param.ResourceId.Split(',');
    query = query.Where(data => resourceId.Contains(data.ResourceId.ToString()));
}
return Ok(query.ToList());
}
}

public class Params
{
    public DateTime? StartDate { get; set; }
    public DateTime? EndDate { get; set; }
    public string ResourceId { get; set; }
}
,
```

* The property will be effective, when large number of resources and appointments bound to the Scheduler.

* This property is applicable only when [resource grouping](#) is enabled in Scheduler.

See Also

- [Virtual scrolling in Agenda view](#)

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Editor template in React Schedule component

Scheduler makes use of popups and dialog to display the required notifications, as well as includes an editor window with event fields for making the appointment creation and editing process easier. You can also easily customize the editor window and the fields present in it, and can also apply validations on those fields.

Event editor

The editor window usually opens on the Scheduler, when a cell or event is double clicked. When a cell is double clicked, the detailed editor window opens in "Add new" mode, whereas when an event is double clicked, the same is opened in an "Edit" mode.

In mobile devices, you can open the detailed editor window in edit mode by clicking the edit icon on the popup, that opens on single tapping an event. You can also open it in add mode by single tapping a cell, which will display a **+** indication, clicking on it again will open the editor window.

You can also prevent the editor window from opening, by rendering Scheduler in a **readonly** mode or by doing code customization within the **popupOpen** event.

How to change the editor window header title and text of footer buttons

You can change the header title and the text of buttons displayed at the footer of the editor window by changing the appropriate localized word collection used in the Scheduler.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, TimelineViews, Month, ViewsDirective,
ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { L10n } from '@syncfusion/ej2-base';
import { scheduleData } from './datasource';
L10n.load({
  'en-US': {
    'schedule': {
      'saveButton': 'Add',
      'cancelButton': 'Close',
      'deleteButton': 'Remove',
      'newEvent': 'Add Event',
    },
  },
});
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews, Month]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, TimelineViews, Month, ViewsDirective,
  ViewDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { L10n } from '@syncfusion/ej2-base';
import { scheduleData } from './datasource';
```

```

L10n.load({
  'en-US': {
    'schedule': {
      'saveButton': 'Add',
      'cancelButton': 'Close',
      'deleteButton': 'Remove',
      'newEvent': 'Add Event',
    },
  },
});
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews, Month]} />
  </ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />

```



```

<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

How to change the label text of default editor fields

To change the default labels such as Subject, Location and other field names in the editor window, make use of the `title` property available within the field option of `eventSettings`.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, TimelineViews, Month, ViewsDirective,
ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const fieldsData = {
    id: 'Id',
    subject: { name: 'Subject', title: 'Event Name' },
    location: { name: 'Location', title: 'Event Location' },
    description: { name: 'Description', title: 'Event Description' },
    startTime: { name: 'StartTime', title: 'Start Duration' },
    endTime: { name: 'EndTime', title: 'End Duration' }
  }
  const eventSettings = { dataSource: scheduleData, fields: fieldsData };
  return (<ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews, Month]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));

```

```
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, TimelineViews, Month, ViewsDirective,
  ViewDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const fieldsData = {
    id: 'Id',
    subject: { name: 'Subject', title: 'Event Name' },
    location: { name: 'Location', title: 'Event Location' },
    description: { name: 'Description', title: 'Event Description' },
    startTime: { name: 'StartTime', title: 'Start Duration' },
    endTime: { name: 'EndTime', title: 'End Duration' }
  }
  const eventSettings: EventSettingsModel = { dataSource: scheduleData,
  fields: fieldsData };
  return (<ScheduleComponent width='100%' height='500px' selectedDate={new
  Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews, Month]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Field validation

It is possible to validate the required fields of the editor window from client-side before submitting it, by adding appropriate validation rules to each field. The appointment fields have been extended to accept both **string** and **object** type values. To perform validations, it is necessary to specify object values for the event fields.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const minValidation = (args) => {
        return args['value'].length >= 5;
    };
    const fieldsData = {
        id: 'Id',
        subject: { name: 'Subject', validation: { required: true } },
        location: { name: 'Location', validation: { required: true } },
        description: {
            name: 'Description', validation: {
                required: true, minLength: [minValidation, 'Need atleast 5
letters to be entered']
            }
        }
    };

```

```

    },
    startTime: { name: 'StartTime', validation: { required: true } },
    endTime: { name: 'EndTime', validation: { required: true } }
  }
  const eventSettings = { dataSource: scheduleData, fields: fieldsData };
  return (<ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const minValidation: (args: { [key: string]: string }) => boolean = (args:
{ [key: string]: string }) => {
    return args['value'].length >= 5;
  };
  const fieldsData = {
    id: 'Id',
    subject: { name: 'Subject', validation: { required: true } },
    location: { name: 'Location', validation: { required: true } },
    description: {
      name: 'Description', validation: {
        required: true, minLength: [minValidation, 'Need atleast 5 letters to
be entered']
      }
    },
  },
  startTime: { name: 'StartTime', validation: { required: true } },
  endTime: { name: 'EndTime', validation: { required: true } }
}
  const eventSettings: EventSettingsModel = { dataSource: scheduleData,
fields: fieldsData };
  return (<ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">

```

```

<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Applicable validation rules can be referred from [form validation](#) documentation.

Add additional fields to the default editor

The additional fields can be added to the default event editor by making use of the `popupOpen` event which gets triggered before the event editor opens on the Scheduler. The `popupOpen` is a client-side event that triggers before any of the generic popups opens on the Scheduler. The additional field (any of the form elements) should be added with a common class name `e-field`, so as to handle and process

those additional data along with the default event object. In the following example, an additional field **Event Type** has been added to the default event editor and its value is processed accordingly.

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, Month, Inject } from '@syncfusion/ej2-react-schedule';
import { createElement } from '@syncfusion/ej2-base';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onPopupOpen = (args) => {
        if (args.type === 'Editor') {
            if (!args.element.querySelector('.custom-field-row')) {
                let row = createElement('div', { className: 'custom-field-
row' });
                let formElement = args.element.querySelector('.e-schedule-
form');
                formElement.firstChild.insertBefore(row,
formElement.firstChild.firstChild);
                let container = createElement('div', { className: 'custom-
field-container' });
                let inputEle = createElement('input', {
                    className: 'e-field', attrs: { name: 'EventType' }
                });
                container.appendChild(inputEle);
                row.appendChild(container);
                let dropdownList = new DropDownList({
                    dataSource: [
                        { text: 'Public Event', value: 'public-event' },
                        { text: 'Maintenance', value: 'maintenance' },
                        { text: 'Commercial Event', value: 'commercial-event' },
                        { text: 'Family Event', value: 'family-event' }
                    ],
                    fields: { text: 'text', value: 'value' },
                    value: args.data.EventType,
                    floatLabelType: 'Always', placeholder: 'Event Type'
                });
                dropdownList.appendTo(inputEle);
                inputEle.setAttribute('name', 'EventType');
            }
        }
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} popupOpen={onPopupOpen}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>);
```

```

}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
    ScheduleComponent, ViewsDirective, ViewDirective, Day, Week, WorkWeek,
    Month, PopupOpenEventArgs, Inject
} from '@syncfusion/ej2-react-schedule';
import { createElement } from '@syncfusion/ej2-base';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onPopupOpen = (args: PopupOpenEventArgs): void => {
        if (args.type === 'Editor') {
            if (!args.element.querySelector('.custom-field-row')) {
                let row: HTMLElement = createElement('div', { className: 'custom-
field-row' });
                let formElement: HTMLElement = args.element.querySelector('.e-
schedule-form');
                formElement.firstChild.insertBefore(row,
formElement.firstChild.firstChild);
                let container: HTMLElement = createElement('div', { className:
'custom-field-container' });
                let inputEle: HTMLInputElement = createElement('input', {
                    className: 'e-field', attrs: { name: 'EventType' }
                }) as HTMLInputElement;
                container.appendChild(inputEle);
                row.appendChild(container);
                let dropdownList: DropDownList = new DropDownList({
                    dataSource: [
                        { text: 'Public Event', value: 'public-event' },
                        { text: 'Maintenance', value: 'maintenance' },
                        { text: 'Commercial Event', value: 'commercial-event' },
                        { text: 'Family Event', value: 'family-event' }
                    ],
                    fields: { text: 'text', value: 'value' },
                    value: (args.data as { [key: string]: Object }).EventType as
string,
                    floatLabelType: 'Always', placeholder: 'Event Type'
                });
                dropdownList.appendTo(inputEle);
                inputEle.setAttribute('name', 'EventType');
            }
        }
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
        eventSettings={eventSettings} popupOpen={onPopupOpen} >
        <ViewsDirective>
            <ViewDirective option='Day' />

```

```

    <ViewDirective option='Week' />
    <ViewDirective option='WorkWeek' />
    <ViewDirective option='Month' />
  </ViewsDirective>
  <Inject services={[Day, Week, WorkWeek, Month]} />
</ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>

```



```

        <div id='schedule'>
            <div id='loader'>Loading....</div>
        </div>
    </body>
</html>

```

Customizing the default time duration in editor window

In default event editor window, start and end time duration are processed based on the **interval** value set within the **timeScale** property. By default, **interval** value is set to 30, and therefore the start/end time duration within the event editor will be in a 30 minutes time difference. You can change this duration value by changing the **duration** option within the **popupOpen** event as shown in the following code example.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, Month, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onPopupOpen = (args) => {
        if (args.type === 'Editor') {
            args.duration = 60;
        }
    }
    return (
        <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} popupOpen={onPopupOpen}>
            <ViewsDirective>
                <ViewDirective option='Day' />
                <ViewDirective option='Week' />
                <ViewDirective option='WorkWeek' />
                <ViewDirective option='Month' />
            </ViewsDirective>
            <Inject services={[Day, Week, WorkWeek, Month]} />
        </ScheduleComponent>
    );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
    ScheduleComponent, ViewsDirective, ViewDirective, Day, Week, WorkWeek,
    Month, PopupOpenEventArgs, Inject
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onPopupOpen = (args: PopupOpenEventArgs): void => {
        if (args.type === 'Editor') {
            args.duration = 60;
        }
    }

```

```

    }
  }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings} popupOpen={onPopupOpen} >
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;

```

```

        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

How to prevent the display of editor and quick popups

It is possible to prevent the display of editor and quick popup windows by passing the value `true` to `cancel` option within the `popupOpen` event.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, Month, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onPopupOpen = (args) => {
        args.cancel = true;
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} popupOpen={onPopupOpen}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
    ScheduleComponent, ViewsDirective, ViewDirective, Day, Week, WorkWeek,
    Month, PopupOpenEventArgs, Inject
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };

```

```

const onPopupOpen = (args: PopupOpenEventArgs): void => {
  args.cancel = true;
}
return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
  eventSettings={eventSettings} popupOpen={onPopupOpen} >
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>)
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdn.jsdelivr.net/npm/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;

```

```

        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

In case, if you need to prevent only specific popups on Scheduler, then you can check the condition based on the popup type. The types of the popup that can be checked within the `popupOpen` event are as follows.

Type	Description
Editor	For Detailed editor window.
QuickInfo	For Quick popup which opens on cell click.
EditEventInfo	For Quick popup which opens on event click.
ViewEventInfo	For Quick popup which opens on responsive mode.
EventContainer	For more event indicator popup.
RecurrenceAlert	For edit recurrence event alert popup.
DeleteAlert	For delete confirmation popup.
ValidationAlert	For validation alert popup.
RecurrenceValidationAlert	For recurrence validation alert popup.

Customizing timezone collection in the editor window

By default, the timezone collections in the editor window have been loaded with built-in timezone collections. Now we can be able to customize the timezone collections using the `timezoneDataSource` property with the collection of `TimezoneFields` data.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
const App = () => {
    const data = [{
        Id: 1,
        Subject: 'Explosion of Betelgeuse Star',
        StartTime: new Date(2020, 1, 15, 10, 0),
        EndTime: new Date(2018, 1, 15, 12, 30),
        IsAllDay: false
    }, {

```

```

        Id: 2,
        Subject: 'Blue Moon Eclipse',
        StartTime: new Date(2020, 1, 16, 12, 0),
        EndTime: new Date(2018, 1, 16, 13, 0),
        IsAllDay: false
    }];
    const eventSettings = { dataSource: data };
    return (<ScheduleComponent height='550px' selectedDate={new Date(2020, 1,
15)} eventSettings={eventSettings} timezoneDataSource=[
        { Value: 'Pacific/Niue', Text: 'Niue' },
        { Value: 'Pacific/Pago_Pago', Text: 'Pago Pago' },
        { Value: 'Pacific/Honolulu', Text: 'Hawaii Time' },
        { Value: 'Pacific/Rarotonga', Text: 'Rarotonga' },
        { Value: 'Pacific/Tahiti', Text: 'Tahiti' },
    ]>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
    const data: object[] = [{
        Id: 1,
        Subject: 'Explosion of Betelgeuse Star',
        StartTime: new Date(2020, 1, 15, 10, 0),
        EndTime: new Date(2018, 1, 15, 12, 30),
        IsAllDay: false
    }, {
        Id: 2,
        Subject: 'Blue Moon Eclipse',
        StartTime: new Date(2020, 1, 16, 12, 0),
        EndTime: new Date(2018, 1, 16, 13, 0),
        IsAllDay: false
    }];
    const eventSettings: EventSettingsModel = { dataSource: data };
    return (<ScheduleComponent height='550px'
        selectedDate={new Date(2020, 1, 15)}
        eventSettings={eventSettings}
        timezoneDataSource=[
            { Value: 'Pacific/Niue', Text: 'Niue' },
            { Value: 'Pacific/Pago_Pago', Text: 'Pago Pago' },
            { Value: 'Pacific/Honolulu', Text: 'Hawaii Time' },
            { Value: 'Pacific/Rarotonga', Text: 'Rarotonga' },
            { Value: 'Pacific/Tahiti', Text: 'Tahiti' },
        ]>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}

```

```

    </ScheduleComponent>)
  };
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Customizing event editor using template

The event editor window can be customized by making use of the `editorTemplate` option. Here, the custom window design is built with the required fields using the script template and its type should be of `text/x-template`.

Each field defined within template should contain the `e-field` class, so as to allow the processing of those field values internally. The ID of this customized script template section is assigned to the `editorTemplate` option, so that these customized fields will be replaced onto the default editor window.

Note: `e-field` class only applicable for **DropDownList**, **DateTimePicker**, **MultiSelect**, **DatePicker**, **CheckBox** and **TextBox** components. Since we have processed the field values internally for the above mentioned components.

As we are using our Syncfusion sub-components within our editor using template in the following example, the custom defined form elements needs to be configured as required Syncfusion components such as **DropDownList** and [Link to the Video](#) within the `popupOpen` event. This particular step can be skipped, if the user needs to simply use the usual form elements.

Learn how to customize the event editor window using templates from this video:

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { DateTimePickerComponent } from '@syncfusion/ej2-react-calendars';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  const onPopupOpen = (args) => {
    if (args.type === 'Editor') {
      let statusElement = args.element.querySelector('#EventType');
      if (statusElement) {
        statusElement.setAttribute('name', 'EventType');
      }
    }
  }
  const editorTemplate = (props) => {
    return (props !== undefined ? <table className="custom-event-editor"
style={{ width: '100%', padding: '5' }}><tbody>
      <tr><td className="e-textlabel">Summary</td><td colspan={4}>
        <input id="Summary" className="e-field e-input" type="text"
name="Subject" style={{ width: '100%' }} />
      </td></tr>
      <tr><td className="e-textlabel">Status</td><td colspan={4}>
        <DropDownListComponent id="EventType" placeholder='Choose status'
data-name="EventType" className="e-field" style={{ width: '100%' }}
dataSource={['New', 'Requested', 'Confirmed']} value={props.EventType ||
null}></DropDownListComponent>
      </td></tr>
      <tr><td className="e-textlabel">From</td><td colspan={4}>

```



```

        <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="StartTime"
data-name="StartTime" value={new Date(props.startTime || props.StartTime)}
className="e-field"></DateTimePickerComponent>
    </td></tr>
    <tr><td className="e-textlabel">To</td><td colSpan={4}>
        <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="EndTime" data-
name="EndTime" value={new Date(props.endTime || props.EndTime)} className="e-
field"></DateTimePickerComponent>
    </td></tr>
    <tr><td className="e-textlabel">Reason</td><td colSpan={4}>
        <textarea id="Description" className="e-field e-input"
name="Description" rows={3} cols={50} style={{ width: '100%', height: '60px
!important', resize: 'vertical' }}></textarea>
    </td></tr></tbody></table> : <div></div>;
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}
editorTemplate={editorTemplate.bind(this)} showQuickInfo={false}
popupOpen={onPopupOpen.bind(this)}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
            <ViewDirective option='Agenda' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
    ScheduleComponent, ViewsDirective, ViewDirective, Day, Week, WorkWeek,
    Month, Agenda, PopupOpenEventArgs, Inject
} from '@syncfusion/ej2-react-schedule';
import { DateTimePickerComponent } from '@syncfusion/ej2-react-calendars';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onPopupOpen = (args: PopupOpenEventArgs): void => {
        if (args.type === 'Editor') {
            let statusElement: HTMLInputElement =
args.element.querySelector('#EventType') as HTMLInputElement;
            if (statusElement) {
                statusElement.setAttribute('name', 'EventType');
            }
        }
    }
    const editorTemplate = (props: Object): JSX.Element => {

```

```

    return (props !== undefined ? <table className="custom-event-editor"
style={{ width: '100%', padding: '5' }}><tbody>
    <tr><td className="e-textlabel">Summary</td><td colspan={4}>
    <input id="Summary" className="e-field e-input" type="text"
name="Subject" style={{ width: '100%' }} />
    </td></tr>
    <tr><td className="e-textlabel">Status</td><td colspan={4}>
    <DropDownListComponent id="EventType" placeholder='Choose status'
data-name="EventType" className="e-field" style={{ width: '100%' }}
dataSource=[{ 'New', 'Requested', 'Confirmed' }] value={ (props as
any).EventType || null }></DropDownListComponent>
    </td></tr>
    <tr><td className="e-textlabel">From</td><td colspan={4}>
    <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="StartTime"
data-name="StartTime" value={new Date((props as any).startTime || (props as
any).StartTime)} className="e-field"></DateTimePickerComponent>
    </td></tr>
    <tr><td className="e-textlabel">To</td><td colspan={4}>
    <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="EndTime" data-
name="EndTime" value={new Date((props as any).endTime || (props as
any).EndTime)} className="e-field"></DateTimePickerComponent>
    </td></tr>
    <tr><td className="e-textlabel">Reason</td><td colspan={4}>
    <textarea id="Description" className="e-field e-input"
name="Description" rows={3} cols={50} style={{ width: '100%', height: '60px
!important', resize: 'vertical' }}></textarea>
    </td></tr></tbody></table> : <div></div>);
  }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings} editorTemplate={editorTemplate.bind(this)}
showQuickInfo={false}
    popupOpen={onPopupOpen.bind(this)} >
    <ViewsDirective>
    <ViewDirective option='Day' />
    <ViewDirective option='Week' />
    <ViewDirective option='WorkWeek' />
    <ViewDirective option='Month' />
    <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>)
  );
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008c00;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

How to customize header and footer using template

The editor window's header and footer can be enhanced with custom designs using the [editorHeaderTemplate](#) and [editorFooterTemplate](#) options. To achieve this, create a script template that includes the necessary fields. Ensure that the template type is set to **text/x-template**.

In this demo, we tailor the editor's header according to the appointment's subject field using the [editorHeaderTemplate](#). Furthermore, we make use of the [editorFooterTemplate](#) to handle the functionality of validating specific fields before proceeding with the save action or canceling it if validation requirements are not met.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';

```

```

import { useRef } from 'react';
import {
    ScheduleComponent, ViewsDirective, ViewDirective, Day, Week, WorkWeek,
    Month, Agenda, PopupOpenEventArgs, Inject
} from '@syncfusion/ej2-react-schedule';
const App = () => {
    const scheduleObj = useRef(null);
    const today = new Date();
    const data = [{
        Id: 1,
        Subject: 'Surgery - Andrew',
        StartTime: new Date(today.getFullYear(), today.getMonth(),
today.getDate(), 9, 0),
        EndTime: new Date(today.getFullYear(), today.getMonth(), today.getDate(),
10, 0),
        IsAllDay: false
    },
    {
        Id: 2,
        Subject: 'Consulting - John',
        StartTime: new Date(today.getFullYear(), today.getMonth(),
today.getDate(), 10, 0),
        EndTime: new Date(today.getFullYear(), today.getMonth(), today.getDate(),
11, 30),
        IsAllDay: false
    },
    {
        Id: 3,
        Subject: 'Therapy - Robert',
        StartTime: new Date(today.getFullYear(), today.getMonth(),
today.getDate(), 11, 30),
        EndTime: new Date(today.getFullYear(), today.getMonth(), today.getDate(),
12, 30),
        IsAllDay: false
    },
    {
        Id: 4,
        Subject: 'Observation - Steven',
        StartTime: new Date(today.getFullYear(), today.getMonth(),
today.getDate(), 12, 30),
        EndTime: new Date(today.getFullYear(), today.getMonth(), today.getDate(),
13, 30),
        IsAllDay: false
    }
    ];
    const eventSettings = { dataSource: data };
    const editorHeaderTemplate = (props) => {
        return (
            <div id="event-header">
                {(props !== undefined) ? ((props.Subject) ?
<div>{props.Subject}</div> : <div>Create New Event</div>) : <div></div>}
            </div>
        );
    }
    const editorFooterTemplate = () => {
        return (
            <div id="event-footer">
                <div id="verify">

```

```

        <input type="checkbox" id="check-box" value="unchecked" />
        <label htmlFor="check-box" id="text">
            Verified
        </label>
    </div>
    <div id="right-button">
        <button id="Save" className="e-control e-btn e-primary" disabled
data-ripple="true">
            Save
        </button>
        <button id="Cancel" className="e-control e-btn e-primary" data-
ripple="true">
            Cancel
        </button>
    </div>
</div>
);
}
const onSaveButtonClick = (args) => {
    const data = {
        Id: args.data.Id,
        Subject: args.element.querySelector('#Subject').value,
        StartTime:
args.element.querySelector('#StartTime').ej2_instances[0].value,
        EndTime: args.element.querySelector('#EndTime').ej2_instances[0].value,
        IsAllDay: args.element.querySelector('#IsAllDay').checked
    };
    if (args.target.classList.contains('e-appointment')) {
        scheduleObj.current.saveEvent(data, 'Save');
    } else {
        data.Id = scheduleObj.current.getEventMaxID();
        scheduleObj.current.addEvent(data);
    }
    scheduleObj.current.closeEditor();
}
const onPopupOpen = (args) => {
    if (args.type === 'Editor') {
        setTimeout(() => {
            const saveButton = args.element.querySelector('#Save');
            const cancelButton = args.element.querySelector('#Cancel');
            const checkBox = args.element.querySelector('#check-box');
            checkBox.onChange = () => {
                if (!checkBox.checked) {
                    saveButton.setAttribute('disabled', '');
                } else {
                    saveButton.removeAttribute('disabled');
                }
            };
            saveButton.onclick = () => {
                onSaveButtonClick(args);
            }
            cancelButton.onclick = () => {
                scheduleObj.current.closeEditor();
            };
        }, 100);
    }
}
}

```

```

return (<ScheduleComponent width='100%' height='550px' ref={scheduleObj}
  eventSettings={eventSettings}
  editorHeaderTemplate={editorHeaderTemplate}
  editorFooterTemplate={editorFooterTemplate}
  popupOpen={onPopupOpen}>
  <ViewsDirective>
    <ViewDirective option='Day' />
    <ViewDirective option='Week' />
    <ViewDirective option='WorkWeek' />
    <ViewDirective option='Month' />
    <ViewDirective option='Agenda' />
  </ViewsDirective>
  <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
</ScheduleComponent>)
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import {
  ScheduleComponent, ViewsDirective, ViewDirective, Day, Week, WorkWeek,
  Month, Agenda, PopupOpenEventArgs, Inject
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const today: Date = new Date();
  const data: Record<string, any>[] = [{
    Id: 1,
    Subject: 'Surgery - Andrew',
    StartTime: new Date(today.getFullYear(), today.getMonth(),
today.getDate(), 9, 0),
    EndTime: new Date(today.getFullYear(), today.getMonth(), today.getDate(),
10, 0),
    IsAllDay: false
  },
  {
    Id: 2,
    Subject: 'Consulting - John',
    StartTime: new Date(today.getFullYear(), today.getMonth(),
today.getDate(), 10, 0),
    EndTime: new Date(today.getFullYear(), today.getMonth(), today.getDate(),
11, 30),
    IsAllDay: false
  },
  {
    Id: 3,
    Subject: 'Therapy - Robert',
    StartTime: new Date(today.getFullYear(), today.getMonth(),
today.getDate(), 11, 30),
    EndTime: new Date(today.getFullYear(), today.getMonth(), today.getDate(),
12, 30),
    IsAllDay: false
  }
];

```

```

    },
    {
      Id: 4,
      Subject: 'Observation - Steven',
      StartTime: new Date(today.getFullYear(), today.getMonth(),
today.getDate(), 12, 30),
      EndTime: new Date(today.getFullYear(), today.getMonth(), today.getDate(),
13, 30),
      IsAllDay: false
    }
  ];
  const eventSettings = { dataSource: data };
  const editorHeaderTemplate = (props: Record<string, any>) => {
    return (
      <div id="event-header">
        {(props !== undefined) ? ((props.Subject) ?
<div>{props.Subject}</div> : <div>Create New Event</div>) : <div></div>}
        </div>
      );
    }
  const editorFooterTemplate = () => {
    return (
      <div id="event-footer">
        <div id="verify">
          <input type="checkbox" id="check-box" value="unchecked" />
          <label htmlFor="check-box" id="text">Verified</label>
        </div>
        <div id="right-button">
          <button id="Save" className="e-control e-btn e-primary" disabled
data-ripple="true">
            Save
          </button>
          <button id="Cancel" className="e-control e-btn e-primary" data-
ripple="true">
            Cancel
          </button>
        </div>
      </div>
    );
  }
  const onSaveButtonClick = (args: PopupOpenEventArgs) => {
    const data: Record<string, any> = {
      Id: args.data.Id,
      Subject: (args.element.querySelector('#Subject') as
HTMLInputElement).value,
      StartTime: (args.element.querySelector('#StartTime') as
any).ej2_instances[0].value,
      EndTime: (args.element.querySelector('#EndTime') as
any).ej2_instances[0].value,
      IsAllDay: (args.element.querySelector('#IsAllDay') as
HTMLInputElement).checked
    };
    if (args.target.classList.contains('e-appointment')) {
      scheduleObj.current.saveEvent(data, 'Save');
    } else {
      data.Id = scheduleObj.current.getEventMaxID();
      scheduleObj.current.addEvent(data);
    }
  }

```

```

    scheduleObj.current.closeEditor();
  }
  const onPopupOpen = (args: PopupOpenEventArgs): void => {
    if (args.type === 'Editor') {
      setTimeout(() => {
        const saveButton: HTMLElement = args.element.querySelector('#Save')
        as HTMLElement;
        const cancelButton: HTMLElement =
args.element.querySelector('#Cancel') as HTMLElement;
        const checkBox: HTMLInputElement =
args.element.querySelector('#check-box') as HTMLInputElement;
        checkBox.onChange = () => {
          if (!(checkBox as HTMLInputElement).checked) {
            saveButton.setAttribute('disabled', '');
          } else {
            saveButton.removeAttribute('disabled');
          }
        };
        saveButton.onclick = () => {
          onSaveButtonClick(args);
        }
        cancelButton.onclick = () => {
          scheduleObj.current.closeEditor();
        };
      }, 100);
    }
  }
  return (<ScheduleComponent width='100%' height='550px' ref={scheduleObj}
    eventSettings={eventSettings}
    editorHeaderTemplate={editorHeaderTemplate}
    editorFooterTemplate={editorFooterTemplate}
    popupOpen={onPopupOpen}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
  );
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />

```



```

<link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
notification/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/23.1.36/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

[How to add resource options within editor template](#)

The resource field can be added within editor template with multiselect control for allow multiple resources.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, Month, ResourcesDirective, ResourceDirective, Inject } from
'@syncfusion/ej2-react-schedule';
import { DateTimePickerComponent } from '@syncfusion/ej2-react-calendars';

```

```

import { MultiSelectComponent } from '@syncfusion/ej2-react-dropdowns';
import { eventData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: eventData };
  const group = { resources: ['Owners'] };
  const ownerData = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ];
  const fields = { text: 'OwnerText', value: 'Id' };
  const editorTemplate = (props) => {
    return (props !== undefined && Object.keys(props).length > 0 ? <table
className="custom-event-editor" style={{ width: '100%', padding: '5'
}}><tbody>
      <tr><td className="e-textlabel">Summary</td><td colSpan={4}>
        <input id="Summary" className="e-field e-input" type="text"
name="Subject" style={{ width: '100%' }} />
      </td></tr>
      <tr><td className="e-textlabel">Owner</td><td colSpan={4}>
        <MultiSelectComponent className="e-field" placeholder='Choose owner'
data-name="OwnerId" dataSource={ownerData} fields={fields}
value={props.OwnerId} />
      </td></tr>
      <tr><td className="e-textlabel">From</td><td colSpan={4}>
        <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="StartTime"
data-name="StartTime" value={new Date(props.startTime || props.StartTime)}
className="e-field"></DateTimePickerComponent>
      </td></tr>
      <tr><td className="e-textlabel">To</td><td colSpan={4}>
        <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="EndTime" data-
name="EndTime" value={new Date(props.endTime || props.EndTime)} className="e-
field"></DateTimePickerComponent>
      </td></tr>
      <tr><td className="e-textlabel">Reason</td><td colSpan={4}>
        <textarea id="Description" className="e-field e-input"
name="Description" rows={3} cols={50} style={{ width: '100%', height: '60px
!important', resize: 'vertical' }}></textarea>
      </td></tr></tbody></table> : <div></div>);
  }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}
editorTemplate={editorTemplate} showQuickInfo={false} group={group}>
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={false} dataSource={ownerData} textField='OwnerText'
idField='Id' allowGroupEdit={false}
colorField='OwnerColor'></ResourceDirective>
    </ResourcesDirective>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>);

```

```

}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
  ScheduleComponent, ViewsDirective, ViewDirective, Day, Week, WorkWeek,
  Month, EventSettingsModel,
  ResourcesDirective, ResourceDirective, Inject
} from '@syncfusion/ej2-react-schedule';
import { DateTimePickerComponent } from '@syncfusion/ej2-react-calendars';
import { MultiSelectComponent } from '@syncfusion/ej2-react-dropdowns';
import { eventData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: eventData };
  const group = { resources: ['Owners'] };
  const ownerData: Object[] = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ];
  const fields: object = { text: 'OwnerText', value: 'Id' };
  const editorTemplate = (props: Object): JSX.Element => {
    return (props !== undefined && Object.keys(props).length > 0 ? <table
className="custom-event-editor" style={{ width: '100%', padding: '5'
}}><tbody>
  <tr><td className="e-textlabel">Summary</td><td colspan={4}>
    <input id="Summary" className="e-field e-input" type="text"
name="Subject" style={{ width: '100%' }} />
  </td></tr>
  <tr><td className="e-textlabel">Owner</td><td colspan={4}>
    <MultiSelectComponent className="e-field" placeholder='Choose owner'
data-name="OwnerId" dataSource={ownerData} fields={fields}
value={props.OwnerId} />
  </td></tr>
  <tr><td className="e-textlabel">From</td><td colspan={4}>
    <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="StartTime"
data-name="StartTime" value={new Date((props as any).startTime || (props as
any).StartTime)} className="e-field"></DateTimePickerComponent>
  </td></tr>
  <tr><td className="e-textlabel">To</td><td colspan={4}>
    <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="EndTime" data-
name="EndTime" value={new Date((props as any).endTime || (props as
any).EndTime)} className="e-field"></DateTimePickerComponent>
  </td></tr>
  <tr><td className="e-textlabel">Reason</td><td colspan={4}>
    <textarea id="Description" className="e-field e-input"
name="Description" rows={3} cols={50}
style={{ width: '100%', height: '60px !important', resize:
'vertical' }}></textarea>
  </td></tr></tbody></table> : <div></div>;
  }
}

```

```

    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings} editorTemplate={editorTemplate}
showQuickInfo={false}
    group={group}>
    <ResourcesDirective>
    <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={false} dataSource={ownerData} textField='OwnerText'
idField='Id' allowGroupEdit={false}
    colorField='OwnerColor'></ResourceDirective>
    </ResourcesDirective>
    <ViewsDirective>
    <ViewDirective option='Day' />
    <ViewDirective option='Week' />
    <ViewDirective option='WorkWeek' />
    <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <link href="index.css" rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>

```

```

<script src="systemjs.config.js"></script>
<style>
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

[How to add recurrence options within editor template](#)

The following code example shows how to add recurrence options within the editor template by importing **RecurrenceEditor**.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, RecurrenceEditorComponent, ViewsDirective,
ViewDirective, Day, Week, WorkWeek, Month, Inject, PopupOpenEventArgs,
EventSettingsModel } from '@syncfusion/ej2-react-schedule';
import { DateTimePickerComponent } from '@syncfusion/ej2-react-calendars';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef(null);
  const recurrObject = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onPopupClose = (args) => {
    if (args.type === 'Editor' && args.data) {
      args.data.RecurrenceRule = recurrObject.current.value;
    }
  }
  const editorTemplate = (props) => {
    return (props !== undefined ? (
      <table className="custom-event-editor" style={{ width: '100%' }}>
        <tbody>
          <tr>
            <td className="e-textlabel">Summary</td>
            <td colspan={4}>
              <input id="Summary" className="e-field e-input" type="text"
name="Subject" style={{ width: '100%' }} />
            </td>
          </tr>
          <tr>
            <td className="e-textlabel">Status</td>

```

```

        <td colspan={4}>
            <DropDownListComponent
                id="EventType"
                placeholder="Choose status"
                data-name="Status"
                className="e-field"
                style={{ width: '100%' }}
                dataSource={['New', 'Requested', 'Confirmed']}
            />
        </td>
    </tr>
    <tr>
        <td className="e-textlabel">From</td>
        <td colspan={4}>
            <DateTimePickerComponent
                format="dd/MM/yy hh:mm a"
                id="StartTime"
                data-name="StartTime"
                value={new Date(props.startTime || props.StartTime)}
                className="e-field"
            />
        </td>
    </tr>
    <tr>
        <td colspan={4}>
            <td className="e-textlabel">To</td>
            <td colspan={4}>
                <DateTimePickerComponent
                    format="dd/MM/yy hh:mm a"
                    id="EndTime"
                    data-name="EndTime"
                    value={new Date(props.endTime || props.EndTime)}
                    className="e-field"
                />
            </td>
        </tr>
    <tr>
        <td colspan={4}>
            <td className="e-textlabel">Recurrence</td>
            <td colspan={4}>
                <RecurrenceEditorComponent ref={recurObject}
id="RecurrenceEditor" />
            </td>
        </tr>
    <tr>
        <td colspan={4}>
            <td className="e-textlabel">Reason</td>
            <td colspan={4}>
                <textarea
                    id="Description"
                    className="e-field e-input"
                    name="Description"
                    rows={3}
                    cols={50}
                    style={{ width: '100%', height: '60px', resize: 'vertical' }}
                />
            </td>
        </tr>
    </tr>
</tbody>
</table>

```

```

    ) : '');
  };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    ref={scheduleObj}
    eventSettings={eventSettings} editorTemplate={editorTemplate}
showQuickInfo={false}
    popupClose={onPopupClose}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, RecurrenceEditorComponent, ViewsDirective,
ViewDirective, Day, Week, WorkWeek, Month, Inject, PopupCloseEventArgs,
EventSettingsModel } from '@syncfusion/ej2-react-schedule';
import { DateTimePickerComponent } from '@syncfusion/ej2-react-calendars';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const recurrObject = useRef<RecurrenceEditorComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const onPopupClose = (args: PopupCloseEventArgs): void => {
    if (args.type === 'Editor' && args.data) {
      args.data.RecurrenceRule = recurrObject.current.value;
    }
  }
  const editorTemplate = (props: any) => {
    return (props !== undefined ? (
      <table className="custom-event-editor" style={{ width: '100%' }}>
        <tbody>
          <tr>
            <td className="e-textlabel">Summary</td>
            <td colspan={4}>
              <input id="Summary" className="e-field e-input" type="text"
name="Subject" style={{ width: '100%' }} />
            </td>
          </tr>
          <tr>
            <td className="e-textlabel">Status</td>
            <td colspan={4}>
              <DropDownListComponent
id="EventType"

```

```

        placeholder="Choose status"
        data-name="Status"
        className="e-field"
        style={{ width: '100%' }}
        dataSource={['New', 'Requested', 'Confirmed']}
      />
    </td>
  </tr>
  <tr>
    <td className="e-textlabel">From</td>
    <td colspan={4}>
      <DateTimePickerComponent
        format="dd/MM/yy hh:mm a"
        id="StartTime"
        data-name="StartTime"
        value={new Date(props.startTime || props.StartTime)}
        className="e-field"
      />
    </td>
  </tr>
  <tr>
    <td className="e-textlabel">To</td>
    <td colspan={4}>
      <DateTimePickerComponent
        format="dd/MM/yy hh:mm a"
        id="EndTime"
        data-name="EndTime"
        value={new Date(props.endTime || props.EndTime)}
        className="e-field"
      />
    </td>
  </tr>
  <tr>
    <td className="e-textlabel">Recurrence</td>
    <td colspan={4}>
      <RecurrenceEditorComponent ref={recurrObject}
id="RecurrenceEditor" />
    </td>
  </tr>
  <tr>
    <td className="e-textlabel">Reason</td>
    <td colspan={4}>
      <textarea
        id="Description"
        className="e-field e-input"
        name="Description"
        rows={3}
        cols={50}
        style={{ width: '100%', height: '60px', resize: 'vertical' }}
      />
    </td>
  </tr>
</tbody>
</table>
) : '';
};

```



```

    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    ref={scheduleObj}
    eventSettings={eventSettings} editorTemplate={editorTemplate}
showQuickInfo={false}
    popupClose={onPopupClose}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>)
  };
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdn.jsdelivr.net/npm/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;

```

```

        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Apply validations on editor template fields

In the following code example, validation has been added to the status field.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, Month, Inject } from '@syncfusion/ej2-react-schedule';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { DateTimePickerComponent } from '@syncfusion/ej2-react-calendars';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onPopupOpen = (args) => {
        if (args.type === 'Editor') {
            let statusElement = args.element.querySelector('#EventType');
            if (statusElement) {
                statusElement.setAttribute('name', 'EventType');
            }
            if (!isNullOrUndefined(document.getElementById("EventType_Error"))) {
                document.getElementById("EventType_Error").style.display = "none";
                document.getElementById("EventType_Error").style.left = "351px";
            }
            let formElement = args.element.querySelector('.e-schedule-form');
            let validator = formElement.ej2_instances[0];
            validator.addRules('EventType', { required: true });
        }
    }
    const onSelect = (args) => {
        if (!isNullOrUndefined(document.getElementById("EventType_Error"))) {
            document.getElementById("EventType_Error").style.display = "none";
        }
    }
    const editorTemplate = (props) => {
        return ((props !== undefined) ? <table className="custom-event-editor"
style={{ width: '100%', padding: '5' }}><tbody>
            <tr><td className="e-textlabel">Summary</td><td colspan={4}>
                <input id="Summary" className="e-field e-input" type="text"
name="Subject" style={{ width: '100%' }} />
            </td></tr>

```

```

        <tr><td className="e-textlabel">Status</td><td colSpan={4}>
            <DropDownListComponent id="EventType" placeholder='Choose status'
            data-name='EventType' className="e-field" style={{ width: '100%' }}
            dataSource={['New', 'Requested', 'Confirmed']}>
                </DropDownListComponent>
            </td></tr>
        <tr><td className="e-textlabel">From</td><td colSpan={4}>
            <DateTimePickerComponent id="StartTime" format='dd/MM/yy hh:mm a'
            data-name="StartTime" value={new Date(props.startTime || props.StartTime)}
            className="e-field"></DateTimePickerComponent>
        </td></tr>
        <tr><td className="e-textlabel">To</td><td colSpan={4}>
            <DateTimePickerComponent id="EndTime" format='dd/MM/yy hh:mm a' data-
            name="EndTime" value={new Date(props.endTime || props.EndTime)} className="e-
            field"></DateTimePickerComponent>
        </td></tr>
        <tr><td className="e-textlabel">Reason</td><td colSpan={4}>
            <textarea id="Description" className="e-field e-input"
            name="Description" rows={3} cols={50} style={{ width: '100%', height: '60px
            !important', resize: 'vertical' }}></textarea>
        </td></tr></tbody></table> : <div></div>;
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
    Date(2018, 1, 15)} eventSettings={eventSettings}
    editorTemplate={editorTemplate} popupOpen={onPopupOpen}
    showQuickInfo={false}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
    ScheduleComponent, ViewsDirective, ViewDirective, Day, Week, WorkWeek,
    Month,
    Inject, PopupOpenEventArgs, EJ2Instance, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { FormValidator } from '@syncfusion/ej2-inputs';
import { DateTimePickerComponent } from '@syncfusion/ej2-react-calendars';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onPopupOpen = (args: PopupOpenEventArgs): void => {

```

```

    if (args.type === 'Editor') {
        let statusElement: HTMLInputElement =
args.element.querySelector('#EventType') as HTMLInputElement;
        if (statusElement) {
            statusElement.setAttribute('name', 'EventType');
        }
        if (!isNullOrUndefined(document.getElementById("EventType_Error"))) {
            document.getElementById("EventType_Error").style.display = "none";
            document.getElementById("EventType_Error").style.left = "351px";
        }
        let formElement: HTMLFormElement = args.element.querySelector('.e-schedule-
form') as HTMLFormElement;
        let validator: FormValidator = ((formElement as
EJ2Instance).ej2_instances[0] as FormValidator);
        validator.addRules('EventType', { required: true });
    }
}
const onSelect = (args: any): void => {
    if (!isNullOrUndefined(document.getElementById("EventType_Error"))) {
        document.getElementById("EventType_Error").style.display = "none";
    }
}
const editorTemplate = (props: Object): JSX.Element => {
    return ((props !== undefined) ? <table className="custom-event-editor"
style={{ width: '100%', padding: '5' }}><tbody>
        <tr><td className="e-textlabel">Summary</td><td colspan={4}>
            <input id="Summary" className="e-field e-input" type="text"
name="Subject" style={{ width: '100%' }} />
        </td></tr>
        <tr><td className="e-textlabel">Status</td><td colspan={4}>
            <DropDownListComponent id="EventType" placeholder='Choose status'
data-name='EventType' className="e-field" style={{ width: '100%' }}
            dataSource={['New', 'Requested', 'Confirmed']>
        </DropDownListComponent>
        </td></tr>
        <tr><td className="e-textlabel">From</td><td colspan={4}>
            <DateTimePickerComponent id="StartTime" format='dd/MM/yy hh:mm a'
data-name="StartTime" value={new Date((props as any).startTime || (props as
any).StartTime)}
            className="e-field"></DateTimePickerComponent>
        </td></tr>
        <tr><td className="e-textlabel">To</td><td colspan={4}>
            <DateTimePickerComponent id="EndTime" format='dd/MM/yy hh:mm a' data-
name="EndTime" value={new Date((props as any).endTime || (props as
any).EndTime)}
            className="e-field"></DateTimePickerComponent>
        </td></tr>
        <tr><td className="e-textlabel">Reason</td><td colspan={4}>
            <textarea id="Description" className="e-field e-input"
name="Description" rows={3} cols={50}
            style={{ width: '100%', height: '60px !important', resize:
'vertical' }}></textarea>
        </td></tr></tbody></table> : <div></div>;
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
        eventSettings={eventSettings}

```

```

    editorTemplate={editorTemplate} popupOpen={onPopupOpen}
    showQuickInfo={false}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>

```

```

    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

How to save the customized event editor using template

The **e-field** class is not added to each field defined within the template, so you should allow to set those field values externally by using the **popupClose** event.

Note: You can allow to retrieve the data only on the **save** and **delete** option. Data cannot be retrieved on the **close** and **cancel** options in the editor window.

The following code example shows how to save the customized event editor using a template by the **popupClose** event.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { DateTimePickerComponent } from '@syncfusion/ej2-react-calendars';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { scheduleData } from './datasource';
const App = () => {
  const startObj = useRef(null);
  const endObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onPopupOpen = (args) => {
    if (args.type === 'Editor') {
      let subjectElement = args.element.querySelector('#Summary');
      if (subjectElement) {
        subjectElement.value = args.data.Subject || "";
      }
      let statusElement = args.element.querySelector('#EventType');
      if (statusElement) {
        statusElement.setAttribute('name', 'EventType');
      }
      let descriptionElement = args.element.querySelector('#Description');
      if (descriptionElement) {
        descriptionElement.value = args.data.Description || "";
      }
    }
  }
  const onPopupClose = (args) => {
    if (args.type === 'Editor' && !isNullOrUndefined(args.data)) {
      let subjectElement = args.element.querySelector('#Summary');
      if (subjectElement) {
        args.data.Subject = subjectElement.value;
      }
    }
  }

```

```

    }
    let statusElement = args.element.querySelector('#EventType');
    if (statusElement) {
        args.data.EventType = statusElement.value;
    }
    args.data.StartTime = startObj.current.value;
    args.data.EndTime = endObj.current.value;
    let descriptionElement = args.element.querySelector('#Description');
    if (descriptionElement) {
        args.data.Description = descriptionElement.value;
    }
}
}

function editorTemplate(props) {
    return (props !== undefined ? <table className="custom-event-editor"
style={{ width: '100%', padding: '5' }}><tbody>
        <tr><td className="e-textlabel">Summary</td><td colspan={4}>
            <input id="Summary" className="e-input" type="text" value=""
name="Subject" style={{ width: '100%' }} />
        </td></tr>
        <tr><td className="e-textlabel">Status</td><td colspan={4}>
            <DropDownListComponent id="EventType" placeholder='Choose status'
data-name="EventType" style={{ width: '100%' }} dataSource={['New',
'Requested', 'Confirmed']} value={props.EventType ||
null}></DropDownListComponent>
        </td></tr>
        <tr><td className="e-textlabel">From</td><td colspan={4}>
            <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="StartTime"
data-name="StartTime" ref={startObj} value={new Date(props.startTime ||
props.StartTime)}></DateTimePickerComponent>
        </td></tr>
        <tr><td className="e-textlabel">To</td><td colspan={4}>
            <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="EndTime" data-
name="EndTime" ref={endObj} value={new Date(props.endTime ||
props.EndTime)}></DateTimePickerComponent>
        </td></tr>
        <tr><td className="e-textlabel">Reason</td><td colspan={4}>
            <textarea id="Description" className="e-input" name="Description"
rows={3} cols={50} style={{ width: '100%', height: '60px !important', resize:
'vertical' }}></textarea>
        </td></tr></tbody></table> : <div></div>);
}

return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}
editorTemplate={editorTemplate} showQuickInfo={false} popupOpen={onPopupOpen}
popupClose={onPopupClose}>
    <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
        <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
</ScheduleComponent>);
}
;

```

```
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import {
    ScheduleComponent, ViewsDirective, ViewDirective, Day, Week, WorkWeek,
    Month, Agenda, PopupOpenEventArgs, EventSettingsModel, PopupCloseEventArgs,
    Inject
} from '@syncfusion/ej2-react-schedule';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { DateTimePickerComponent } from '@syncfusion/ej2-react-calendars';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { scheduleData } from './datasource';
const App = () => {
    const startObj = useRef(null);
    const endObj = useRef(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onPopupOpen = (args: PopupOpenEventArgs): void => {
        if (args.type === 'Editor') {
            let subjectElement: HTMLInputElement =
args.element.querySelector('#Summary') as HTMLInputElement;
            if (subjectElement) {
                subjectElement.value = ((args.data as { [key: string]: Object
})).Subject as string || "";
            }
            let statusElement: HTMLInputElement =
args.element.querySelector('#EventType') as HTMLInputElement;
            if (statusElement) {
                statusElement.setAttribute('name', 'EventType');
            }
            let descriptionElement: HTMLInputElement =
args.element.querySelector('#Description') as HTMLInputElement;
            if (descriptionElement) {
                descriptionElement.value = ((args.data as { [key: string]: Object
})).Description as string || "";
            }
        }
    }
    const onPopupClose = (args: PopupCloseEventArgs): void => {
        if (args.type === 'Editor' && !isNullOrUndefined(args.data as { [key:
string]: Object })) {
            let subjectElement: HTMLInputElement =
args.element.querySelector('#Summary') as HTMLInputElement;
            if (subjectElement) {
                ((args.data as { [key: string]: Object })).Subject =
subjectElement.value;
            }
            let statusElement: HTMLInputElement =
args.element.querySelector('#EventType') as HTMLInputElement;
            if (statusElement) {
                ((args.data as { [key: string]: Object })).EventType =
statusElement.value;
            }
        }
    }
}
```



```

    }
    ((args.data as { [key: string]: Object })).StartTime =
startObj.current.value;
    ((args.data as { [key: string]: Object })).EndTime =
endObj.current.value;
    let descriptionElement: HTMLInputElement =
args.element.querySelector('#Description') as HTMLInputElement;
    if (descriptionElement) {
        ((args.data as { [key: string]: Object })).Description =
descriptionElement.value;
    }
}
}

const editorTemplate = (props: Object): JSX.Element => {
    return (props !== undefined ? <table className="custom-event-editor"
style={{ width: '100%', padding: '5' }}><tbody>
        <tr><td className="e-textlabel">Summary</td><td colspan={4}>
            <input id="Summary" className="e-input" type="text" value=""
name="Subject" style={{ width: '100%' }} />
        </td></tr>
        <tr><td className="e-textlabel">Status</td><td colspan={4}>
            <DropDownListComponent id="EventType" placeholder='Choose status'
data-name="EventType" style={{ width: '100%' }} dataSource={['New',
'Requested', 'Confirmed']} value={(props as any).EventType ||
null}></DropDownListComponent>
        </td></tr>
        <tr><td className="e-textlabel">From</td><td colspan={4}>
            <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="StartTime"
data-name="StartTime" ref={startObj} value={new Date((props as any).startTime
|| (props as any).StartTime)}></DateTimePickerComponent>
        </td></tr>
        <tr><td className="e-textlabel">To</td><td colspan={4}>
            <DateTimePickerComponent format='dd/MM/yy hh:mm a' id="EndTime" data-
name="EndTime" ref={endObj} value={new Date((props as any).endTime || (props
as any).EndTime)}></DateTimePickerComponent>
        </td></tr>
        <tr><td className="e-textlabel">Reason</td><td colspan={4}>
            <textarea id="Description" className="e-input" name="Description"
rows={3} cols={50} style={{ width: '100%', height: '60px !important', resize:
'vertical' }}></textarea>
        </td></tr></tbody></table> : <div></div>);
}

return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings} editorTemplate={editorTemplate}
showQuickInfo={false}
    popupOpen={onPopupOpen} popupClose={onPopupClose} >
    <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
        <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
</ScheduleComponent>)
};

```

```
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

In case, if you need to prevent only specific popups on Scheduler, then you can check the condition based on the popup type. The types of the popup that can be checked within the `popupClose` event are as follows.

Type	Description
Editor	For Detailed editor window.
QuickInfo	For Quick popup which opens on cell click.
EditEventInfo	For Quick popup which opens on event click.
ViewEventInfo	For Quick popup which opens on responsive mode.
EventContainer	For more event indicator popup.
RecurrenceAlert	For edit recurrence event alert popup.
DeleteAlert	For delete confirmation popup.
ValidationAlert	For validation alert popup.
RecurrenceValidationAlert	For recurrence validation alert popup.

Quick popups

The quick info popups are the ones that gets opened, when a cell or appointment is single clicked on the desktop mode. On single clicking a cell, you can simply provide a subject and save it. Also, while single clicking on an event, a popup will be displayed where you can get the overview of the event information. You can also edit or delete those events through the options available in it.

By default, these popups are displayed over cells and appointments of Scheduler and to disable this action, set `false` to `showQuickInfo` property.

The quick popup that opens while single clicking on the cells are not applicable on mobile devices.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, TimelineViews, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (
    <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} showQuickInfo={false} eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
      </ViewsDirective>
      <Inject services={[Day, TimelineViews, Week, WorkWeek]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
```

```
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import {
  ScheduleComponent, Day, Week, WorkWeek, TimelineViews, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} showQuickInfo={false} eventSettings={eventSettings} >
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
    </ViewsDirective>
    <Inject services={[Day, TimelineViews, Week, WorkWeek]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
```

```

<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

How to open QuickInfo popup on multiple cell selection

By default the **QuickInfo** popup will open on single click of the cell. To open the quick info popup on multiple cell selection, you need to select the cells and press enter key. You can open this popup immediately after multiple cell selection by setting up **true** to **quickInfoOnSelectionEnd** property where as its default value is **false**.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, TimelineViews, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
const App = () => {
    return (<ScheduleComponent width='100%' height='550px'
quickInfoOnSelectionEnd={true}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='TimelineWeek' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
        </ViewsDirective>
        <Inject services={[Day, TimelineViews, Week, WorkWeek]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```
import {
  ScheduleComponent, Day, Week, WorkWeek, TimelineViews, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  return (<ScheduleComponent width='100%' height='550px'
    quickInfoOnSelectionEnd={true} >
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
    </ViewsDirective>
    <Inject services={[Day, TimelineViews, Week, WorkWeek]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
```

```

        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

How to change the watermark text of quick popup subject

By default, **Add Title** text is displayed on the subject field of quick popup. To change the default watermark text, change the value of the appropriate localized word collection used in the Scheduler.

```

`ts
L10n.load({
'en-US': {
'schedule': {
'addTitle': 'New Title'
}
}
});
`

```

Customizing quick popups

The look and feel of the built-in quick popup window, which opens when single clicked on the cells or appointments can be customized by making use of the **quickInfoTemplates** property of the Scheduler. There are 3 sub-options available to customize them easily,

- header - Accepts the template design that customizes the header part of the quick popup.
- content - Accepts the template design that customizes the content part of the quick popup.
- footer - Accepts the template design that customizes the footer part of the quick popup.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { isNullOrUndefined } from "@syncfusion/ej2-base";
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
Resize, DragAndDrop } from "@syncfusion/ej2-react-schedule";
import { scheduleData } from './datasource';
const App = () => {
    const scheduleObj = useRef(null);

```

```

const eventSettings = { dataSource: scheduleData };
const buttonClickActions = (e) => {
  let eventData = {};
  let actionType = "Add";
  const action = e.target.id;
  const getSlotData = () => {
    const cellDetails =
scheduleObj.current.getCellDetails(scheduleObj.current.getSelectedElements())
;
    const eventData =
scheduleObj.current.eventWindow.getObjectFromFormData("e-quick-popup-
wrapper");
    const addObj = {};
    addObj.Id = scheduleObj.current.getEventMaxID();
    addObj.Subject = eventData.Subject.length > 0 ? eventData.Subject :
"Add title";
    addObj.StartTime = new Date(+cellDetails.startTime);
    addObj.EndTime = new Date(+cellDetails.endTime);
    addObj.Location = eventData.Location;
    return addObj;
  };
  switch (action) {
    case "add":
      eventData = getSlotData();
      scheduleObj.current.addEvent(eventData);
      break;
    case "edit":
    case "edit-series":
      eventData = scheduleObj.current.activeEventData.event;
      actionType = eventData.RecurrenceRule ? action === "edit" ?
"EditOccurrence" : "EditSeries" : "Save";
      if (actionType === "EditSeries")
        eventData = scheduleObj.current.eventBase.getParentEvent(eventData,
true);
      scheduleObj.current.openEditor(eventData, actionType);
      break;
    case "delete":
    case "delete-series":
      eventData = scheduleObj.current.activeEventData.event;
      actionType = eventData.RecurrenceRule ? action === "delete" ?
"DeleteOccurrence" : "DeleteSeries" : "Delete";
      if (actionType === "DeleteSeries")
        eventData = scheduleObj.current.eventBase.getParentEvent(eventData,
true);
      scheduleObj.current.deleteEvent(eventData, actionType);
      break;
    case "more-details":
      eventData = getSlotData();
      scheduleObj.current.openEditor(eventData, "Add", true);
      break;
    default:
      break;
  }
  scheduleObj.current.closeQuickInfoPopup();
}
const header = (props) => {
  return (

```



```

    <div>
      {props.elementType === "cell" ? (
        <div className="e-cell-header e-popup-header">
          <div className="e-header-icon-wrapper">
            <button id="close" className="e-close e-close-icon e-icons"
title="Close" onClick={buttonClickActions.bind(this)} />
          </div>
        </div>
      ) : (
        <div className="e-event-header e-popup-header">
          <div className="e-header-icon-wrapper">
            <button id="close" className="e-close e-close-icon e-icons"
title="CLOSE" onClick={buttonClickActions.bind(this)} />
          </div>
        </div>
      )}
    </div>
  );
}
const content = (props) => {
  return (
    <div>
      {props.elementType === "cell" ? (
        <div className="e-cell-content e-template">
          <form className="e-schedule-form">
            <div>
              <input className="subject e-field e-input" type="text"
name="Subject" placeholder="Title" />
            </div>
            <div>
              <input className="location e-field e-input" type="text"
name="Location" placeholder="Location" />
            </div>
          </form>
        </div>
      ) : (
        <div className="e-event-content e-template">
          <div className="e-subject-wrap">
            {props.Subject !== undefined ? (
              <div className="subject">{props.Subject}</div>
            ) : (
              ""
            )}
            {props.Location !== undefined ? (
              <div className="location">{props.Location}</div>
            ) : (
              ""
            )}
            {props.Description !== undefined ? (
              <div className="description">{props.Description}</div>
            ) : (
              ""
            )}
          </div>
        </div>
      )}
    </div>
  );
}
</div>

```

```

    );
  }
  const footer = (props) => {
    return (
      <div>
        {props.elementType === "cell" ? (
          <div className="e-cell-footer">
            <div className="left-button">
              <button id="more-details" className="e-event-details"
title="Extra Details" onClick={buttonClickActions.bind(this)}> Extra Details
</button>
            </div>
            <div className="right-button">
              <button id="add" className="e-event-create" title="Add"
onClick={buttonClickActions.bind(this)} > Add </button>
            </div>
          </div>
        ) : (
          <div className="e-event-footer">
            <div className="left-button">
              <button id="edit" className="e-event-edit" title="Edit"
onClick={buttonClickActions.bind(this)} > Edit </button>
              {!isNullOrUndefined(props.RecurrenceRule) &&
                props.RecurrenceRule !== "" ? (
                  <button id="edit-series" className="e-edit-series"
title="Edit Series" onClick={buttonClickActions.bind(this)}> Edit Series
</button>
                ) : (
                  ""
                )}
            </div>
            <div className="right-button">
              <button id="delete" className="e-event-delete" title="Delete"
onClick={buttonClickActions.bind(this)} > Delete </button>
              {!isNullOrUndefined(props.RecurrenceRule) &&
                props.RecurrenceRule !== "" ? (
                  <button id="delete-series" className="e-delete-series"
title="Delete Series" onClick={buttonClickActions.bind(this)}> Delete Series
</button>
                ) : (
                  ""
                )}
            </div>
          </div>
        )}
      </div>
    );
  }
  const quickInfoTemplates = { header: header.bind(this), content:
content.bind(this), footer: footer.bind(this) };
  return (<ScheduleComponent id="schedule" ref={scheduleObj} width="100%"
height="550px" selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings} quickInfoTemplates={quickInfoTemplates}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, Resize,
DragAndDrop]} />
  </ScheduleComponent>)
};

```

```
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { isNullOrUndefined } from "@syncfusion/ej2-base";
import { ScheduleComponent, Day, Week, WorkWeek, Month, EventSettingsModel,
Agenda, Inject, CellClickEventArgs, CurrentAction, Resize, DragAndDrop } from
"@syncfusion/ej2-react-schedule";
import { scheduleData } from './datasource';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const buttonClickActions = (e: Event): void => {
        let eventData: { [key: string]: Object } = {};
        let actionType: CurrentAction = "Add";
        const action: string = (e.target as HTMLElement).id;
        const getSlotData: Function = (): { [key: string]: Object } => {
            const cellDetails: CellClickEventArgs =
scheduleObj.current.getCellDetails(scheduleObj.current.getSelectedElements())
;
            const eventData: { [key: string]: Object; } =
scheduleObj.current.eventWindow.getObjectFromFormData("e-quick-popup-
wrapper");
            const addObj: { [key: string]: Object } = {};
            addObj.Id = scheduleObj.current.getEventMaxID();
            addObj.Subject = (eventData.Subject as string).length > 0 ?
eventData.Subject : "Add title";
            addObj.StartTime = new Date(+cellDetails.startTime);
            addObj.EndTime = new Date(+cellDetails.endTime);
            addObj.Location = eventData.Location;
            return addObj;
        };
        switch (action) {
            case "add":
                eventData = getSlotData();
                scheduleObj.current.addEvent(eventData);
                break;
            case "edit":
            case "edit-series":
                eventData = scheduleObj.current.activeEventData.event as { [key:
string]: Object; };
                actionType = eventData.RecurrenceRule ? action === "edit" ?
"EditOccurrence" : "EditSeries" : "Save";
                if (actionType === "EditSeries")
                    eventData = scheduleObj.current.eventBase.getParentEvent(eventData,
true);
                scheduleObj.current.openEditor(eventData, actionType);
                break;
            case "delete":
            case "delete-series":
                eventData = scheduleObj.current.activeEventData.event as { [key:
string]: Object; };
```

```

        actionType = eventData.RecurrenceRule ? action === "delete" ?
        "DeleteOccurrence" : "DeleteSeries" : "Delete";
        if (actionType === "DeleteSeries")
            eventData = scheduleObj.current.eventBase.getParentEvent(eventData,
true);
        scheduleObj.current.deleteEvent(eventData, actionType);
        break;
        case "more-details":
            eventData = getSlotData();
            scheduleObj.current.openEditor(eventData, "Add", true);
            break;
        default:
            break;
    }
    scheduleObj.current.closeQuickInfoPopup();
}
const header = (props: { [key: string]: string }): JSX.Element => {
    return (
        <div>
            {props.elementType === "cell" ? (
                <div className="e-cell-header e-popup-header">
                    <div className="e-header-icon-wrapper">
                        <button id="close" className="e-close e-close-icon e-icons"
title="Close" onClick={buttonClickActions.bind(this)} />
                    </div>
                </div>
            ) : (
                <div className="e-event-header e-popup-header">
                    <div className="e-header-icon-wrapper">
                        <button id="close" className="e-close e-close-icon e-icons"
title="CLOSE" onClick={buttonClickActions.bind(this)} />
                    </div>
                </div>
            )}
        </div>
    );
}
const content = (props: { [key: string]: string }): JSX.Element => {
    return (
        <div>
            {props.elementType === "cell" ? (
                <div className="e-cell-content e-template">
                    <form className="e-schedule-form">
                        <div>
                            <input className="subject e-field e-input" type="text"
name="Subject" placeholder="Title" />
                        </div>
                        <div>
                            <input className="location e-field e-input" type="text"
name="Location" placeholder="Location" />
                        </div>
                    </form>
                </div>
            ) : (
                <div className="e-event-content e-template">
                    <div className="e-subject-wrap">
                        {props.Subject !== undefined ? (

```

```

        <div className="subject">{props.Subject}</div>
      ) : (
        ""
      )}
      {props.Location !== undefined ? (
        <div className="location">{props.Location}</div>
      ) : (
        ""
      )}
      {props.Description !== undefined ? (
        <div className="description">{props.Description}</div>
      ) : (
        ""
      )}
    </div>
  </div>
)}
</div>
);
}
const footer = (props: { [key: string]: string }): JSX.Element => {
  return (
    <div>
      {props.elementType === "cell" ? (
        <div className="e-cell-footer">
          <div className="left-button">
            <button id="more-details" className="e-event-details"
title="Extra Details" onClick={buttonClickActions.bind(this)}> Extra Details
</button>
          </div>
          <div className="right-button">
            <button id="add" className="e-event-create" title="Add"
onClick={buttonClickActions.bind(this)} > Add </button>
          </div>
        </div>
      ) : (
        <div className="e-event-footer">
          <div className="left-button">
            <button id="edit" className="e-event-edit" title="Edit"
onClick={buttonClickActions.bind(this)} > Edit </button>
            {!isNullOrUndefined(props.RecurrenceRule) &&
              props.RecurrenceRule !== "" ? (
                <button id="edit-series" className="e-edit-series"
title="Edit Series" onClick={buttonClickActions.bind(this)}> Edit Series
</button>
              ) : (
                ""
              )}
          </div>
          <div className="right-button">
            <button id="delete" className="e-event-delete" title="Delete"
onClick={buttonClickActions.bind(this)} > Delete </button>
            {!isNullOrUndefined(props.RecurrenceRule) &&
              props.RecurrenceRule !== "" ? (
                <button id="delete-series" className="e-delete-series"
title="Delete Series" onClick={buttonClickActions.bind(this)}> Delete Series
</button>
              ) : (
                ""
              )}
          </div>
        </div>
      ) : (
        ""
      )}
    </div>
  );
}

```

```

        ) : (
            ""
        )}
    </div>
</div>
)}
</div>
);
}
const quickInfoTemplates = { header: header.bind(this), content:
content.bind(this), footer: footer.bind(this) };
return (<ScheduleComponent id="schedule" ref={scheduleObj} width="100%"
height="550px" selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings} quickInfoTemplates={quickInfoTemplates}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, Resize,
DragAndDrop]} />
</ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdn.jsdelivr.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <link href="index.css" rel="stylesheet" />
    <script src="systemjs.config.js"></script>
</style>

```

```

        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

The quick popup in adaptive mode can also be customized using `quickInfoTemplates` using `e-device` class.

More events indicator and popup

When the number of appointments count that lies on a particular time range * default appointment height exceeds the default height of a cell in month view and all other timeline views, a `+ more` text indicator will be displayed at the bottom of those cells. This indicator denotes that the cell contains few more appointments in it and clicking on that will display a popup displaying all the appointments present on that day.

To disable this option of showing popup with all hidden appointments, while clicking on the text indicator, you can do code customization within the `popupOpen` event.

The same indicator is displayed on all-day row in calendar views such as day, week and work week views alone, when the number of appointment count present in a cell exceeds three. Clicking on the text indicator here will not open a popup, but will allow the expand/collapse option for viewing the remaining appointments present in the all-day row.

The following code example shows how to disable the display of such popups while clicking on the more text indicator.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Month, Inject }
from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onPopupOpen = (args) => {
        if (args.type === 'EventContainer') {
            args.cancel = true;
        }
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
    Date(2018, 1, 15)} currentView='Month' eventSettings={eventSettings}
    popupOpen={onPopupOpen}>

```

```

    <ViewsDirective>
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Month]} />
  </ScheduleComponent>);
}

;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
  ScheduleComponent, ViewsDirective, ViewDirective, Month,
  PopupOpenEventArgs, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const onPopupOpen = (args: PopupOpenEventArgs): void => {
    if (args.type === 'EventContainer') {
      args.cancel = true;
    }
  }
  return (
    <ScheduleComponent width='100%' height='550px' selectedDate={new
    Date(2018, 1, 15)}
      currentView='Month'
      eventSettings={eventSettings} popupOpen={onPopupOpen} >
      <ViewsDirective>
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Month]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
  base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
  buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
  calendars/styles/material.css" rel="stylesheet" />

```



```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

[How to customize the popup that opens on more indicator](#)

The following code example shows you how to customize the default more indicator popup in which number of events rendered count on the day has been shown in the header.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Month, Inject }
from '@syncfusion/ej2-react-schedule';
import { Internationalization } from '@syncfusion/ej2-base';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onPopupOpen = (args) => {
        if (args.type === 'EventContainer') {
            let instance = new Internationalization();
            let date = instance.formatDate(args.data.date, { skeleton: 'MMMEd' });
            (args.element.querySelector('.e-header-date')).innerText = date;
        }
    };

```

```

        (args.element.querySelector('.e-header-day')).innerText = 'Event count: ' + args.data.event.length;
    }
}
return (<ScheduleComponent width='100%' height='550px' selectedDate={new Date(2018, 1, 15)} currentView='Month' eventSettings={eventSettings} popupOpen={onPopupOpen}>
    <ViewsDirective>
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Month]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
    ScheduleComponent, ViewsDirective, ViewDirective, Month,
    PopupOpenEventArgs, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { Internationalization } from '@syncfusion/ej2-base';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onPopupOpen = (args: PopupOpenEventArgs): void => {
        if (args.type === 'EventContainer') {
            let instance: Internationalization = new Internationalization();
            let date: string = instance.formatDate(args.data.date, { skeleton: 'MMMd' });
            ((args.element.querySelector('.e-header-date')) as HTMLElement).innerText = date;
            ((args.element.querySelector('.e-header-day')) as HTMLElement).innerText = 'Event count: ' + args.data.event.length;
        }
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new Date(2018, 1, 15)}
        currentView='Month'
        eventSettings={eventSettings} popupOpen={onPopupOpen} >
        <ViewsDirective>
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Month]} />
    </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```
<!DOCTYPE html>
```

```

<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

How to prevent the display of popup when clicking on the more text indicator

It is possible to prevent the display of popup window by passing the value `true` to `cancel` option within the `MoreEventsClick` event.

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
```

```
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Month, Inject }
from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  const onMoreEventsClick = (args) => {
    args.cancel = true;
  }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} currentView='Month' eventSettings={eventSettings}
moreEventsClick={onMoreEventsClick}>
    <ViewsDirective>
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Month]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
  ScheduleComponent, ViewsDirective, ViewDirective, Month,
  MoreEventsClickArgs, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const onMoreEventsClick = (args: MoreEventsClickArgs): void => {
    args.cancel = true;
  }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    currentView='Month'
    eventSettings={eventSettings} moreEventsClick={onMoreEventsClick} >
    <ViewsDirective>
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Month]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta name="description" content="Essential JS 2 for React Components" />
<meta name="author" content="Syncfusion" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

How to navigate Day view when clicking on more text indicator

The following code example shows you how to customize the `moreEventsClick` property to navigate to the Day view when clicking on the more text indicator.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Month, Inject
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';

```

```

const App = () => {
  const eventSettings = { dataSource: scheduleData };
  const onMoreEventsClick = (args) => {
    args.isPopupOpen = false;
  }
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} currentView='Month' eventSettings={eventSettings}
moreEventsClick={onMoreEventsClick}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Month]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
  ScheduleComponent, ViewsDirective, ViewDirective, Day, Month,
  MoreEventsClickArgs, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const onMoreEventsClick = (args: MoreEventsClickArgs): void => {
    args.isPopupOpen = false;
  }
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    currentView='Month'
    eventSettings={eventSettings} moreEventsClick={onMoreEventsClick} >
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Month]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />

```

```

    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <link href="index.css" rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

How to close the editor window manually

You can close the editor window by using [closeEditor](#) method.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
    const scheduleObj = useRef(null);

```

```

const ClickButton = () => {
    scheduleObj.current.closeEditor();
}
const data = [{
    Id: 1,
    Subject: 'Review Meeting',
    StartTime: new Date(2023, 2, 5, 9, 0, 0),
    EndTime: new Date(2023, 2, 5, 10, 0, 0)
}];
const eventSettings = { dataSource: data };
return (<div>
    <ButtonComponent onClick={ClickButton}>Close Editor Window
</ButtonComponent>
    <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2023, 2, 5)} ref={scheduleObj} currentView='Month'
eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
</div>);
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
ViewDirective, EventSettingsModel, Inject } from '@syncfusion/ej2-react-
schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const ClickButton = () => {
        scheduleObj.current.closeEditor();
    }
    const data: object[] = [{
        Id: 1,
        Subject: 'Review Meeting',
        StartTime: new Date(2023, 2, 5, 9, 0, 0),
        EndTime: new Date(2023, 2, 5, 10, 0, 0)
    }];
    const eventSettings: EventSettingsModel = { dataSource: data };
    return (<div>
        <ButtonComponent onClick={ClickButton}>Close Editor Window
    </ButtonComponent>
        <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2023, 2, 5)}
            ref={scheduleObj} currentView='Month'

```



```

    eventSettings={eventSettings} >
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>
</div>
);
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
    }
  </style>

```

```

        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

How to open the quick info popup manually

You can open the quick info popup in scheduler by using the [openQuickInfoPopup](#) public method. To open the cell quick info popup, you can pass the cell data as an argument to the method. To open the event quick info popup, you should pass the event data object as an argument to the method.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
    const scheduleObj = useRef(null);
    const onCellClickButton = () => {
        let cellData = {
            Subject: 'Review Meeting',
            StartTime: new Date(2023, 2, 5, 9, 0, 0),
            EndTime: new Date(2023, 2, 5, 10, 0, 0)
        };
        scheduleObj.current.openQuickInfoPopup(cellData, 'Add');
    }
    const onEventClickButton = () => {
        let eventData = {
            Id: 1,
            Subject: 'Review Meeting',
            StartTime: new Date(2023, 2, 5, 9, 0, 0),
            EndTime: new Date(2023, 2, 5, 10, 0, 0)
        };
        scheduleObj.current.openQuickInfoPopup(eventData, 'Save');
    }
    const data = [{
        Id: 1,
        Subject: 'Review Meeting',
        StartTime: new Date(2023, 2, 5, 9, 0, 0),
        EndTime: new Date(2023, 2, 5, 10, 0, 0)
    }];
    const eventSettings = { dataSource: data };
    return (<div>
        <ButtonComponent id='btn1' onClick={onCellClickButton}>Show Cell Click
        Popup </ButtonComponent>
        <ButtonComponent id='btn2' onClick={onEventClickButton}>Show Event Click
        Popup </ButtonComponent>
    </div>);
}

```

```

    <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2023, 2, 5)} ref={scheduleObj} currentView='Month'
eventSettings={eventSettings}>
    <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
</ScheduleComponent>
</div>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
ViewDirective, EventSettingsModel, Inject } from '@syncfusion/ej2-react-
schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const onCellClickButton = () => {
        const cellData = {
            Subject: 'Review Meeting',
            StartTime: new Date(2023, 2, 5, 9, 0, 0),
            EndTime: new Date(2023, 2, 5, 10, 0, 0)
        };
        scheduleObj.current.openQuickInfoPopup(cellData, 'Add');
    };
    const onEventClickButton = () => {
        const eventData = {
            Id: 1,
            Subject: 'Review Meeting',
            StartTime: new Date(2023, 2, 5, 9, 0, 0),
            EndTime: new Date(2023, 2, 5, 10, 0, 0)
        };
        scheduleObj.current.openQuickInfoPopup(eventData, 'Save');
    };
    const data = [{
        Id: 1,
        Subject: 'Review Meeting',
        StartTime: new Date(2023, 2, 5, 9, 0, 0),
        EndTime: new Date(2023, 2, 5, 10, 0, 0)
    }];
    const eventSettings: EventSettingsModel = { dataSource: data };
    return (
        <div>
            <ButtonComponent id='btn1' onClick={onCellClickButton}>Show Cell Click
            Popup </ButtonComponent>

```

```

    <ButtonComponent id='btn2' onClick={onEventClickButton}>Show Event
Click Popup </ButtonComponent>
    <ScheduleComponent
        width='100%'
        height='550px'
        selectedDate={new Date(2023, 2, 5)}
        ref={scheduleObj}
        currentView='Month'
        eventSettings={eventSettings}
    >
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
</div>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <link href="index.css" rel="stylesheet" />

```

```

<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

How to close the quick info popup manually

You can close the quick info popup in scheduler by using the [closeQuickInfoPopup](#) public method. The following code example demonstrates the how to close quick info popup manually.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef(null);
  const ClickButton = () => {
    scheduleObj.current.closeQuickInfoPopup();
  }
  const data = [{
    Id: 1,
    Subject: 'Review Meeting',
    StartTime: new Date(2023, 2, 5, 9, 0, 0),
    EndTime: new Date(2023, 2, 5, 10, 0, 0)
  }];
  const eventSettings = { dataSource: data };
  return (<div>
    <ButtonComponent onClick={ClickButton.bind(this)}>Close QuickInfo
    Popup</ButtonComponent>
    <ScheduleComponent width='100%' height='550px' selectedDate={new
    Date(2023, 2, 5)} ref={scheduleObj} currentView='Month'
    eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />

```

```

        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
  </div>);
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
ViewDirective, EventSettingsModel, Inject } from '@syncfusion/ej2-react-
schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const ClickButton = () => {
    scheduleObj.current.closeQuickInfoPopup();
  }
  const data: object[] = [{
    Id: 1,
    Subject: 'Review Meeting',
    StartTime: new Date(2023, 2, 5, 9, 0, 0),
    EndTime: new Date(2023, 2, 5, 10, 0, 0)
  }];
  const eventSettings: EventSettingsModel = { dataSource: data };
  return (<div>
    <ButtonComponent onClick={ClickButton.bind(this)}>Close QuickInfo
Popup</ButtonComponent>
    <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2023, 2, 5)}
      ref={scheduleObj} currentView='Month'
      eventSettings={eventSettings} >
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
  </div>
  );
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>

```

```

<title>Syncfusion React Schedule</title>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta name="description" content="Essential JS 2 for React Components" />
<meta name="author" content="Syncfusion" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Timezone in React Schedule component

The Scheduler makes use of the current system time zone by default. If it needs to follow some other user-specific time zone, then the `timezone` property needs to be used. Apart from the default action of applying specific timezone to the Scheduler, it is also possible to set different time zone values for each

appointments through the properties `startTimezone` and `endTimezone` which can be defined as separate fields within the event fields collection.

Note: **timezone** property only applicable for the appointment processing and current time indication.

Understanding date manipulation in JavaScript

The `new Date()` in JavaScript returns the exact current date object with complete time and timezone information. For example, it may return value such as `Wed Dec 12 2018 05:23:27 GMT+0530 (India Standard Time)` which indicates that the current date is December 12, 2018 and the current time is 5.23 AM on browsers following the IST timezone.

Scheduler with no timezone

When no specific time zone is set to Scheduler, appointments will be displayed based on the client system's timezone which is the default behavior. Here, the same appointment when viewed from different timezone will have different start and end times.

The following code example displays an appointment from 9.00 AM to 10.00 AM when you open the Scheduler from any of the timezone. This is because, we are providing the start and end time enclosing with `new Date()` which works based on the client browser's timezone.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
const App = () => {
  const scheduleData = [{
    Id: 3,
    Subject: 'Paris',
    StartTime: new Date(2018, 1, 15, 9, 0),
    EndTime: new Date(2018, 1, 15, 10, 0)
  }];
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new Date(2018, 1, 11)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const scheduleData: Object[] = [{
    Id: 3,
    Subject: 'Paris',
```



```

    StartTime: new Date(2018, 1, 15, 9, 0),
    EndTime: new Date(2018, 1, 15, 10, 0)
  }];
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (
    <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 11)} eventSettings={eventSettings} >
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>

```

```

    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Scheduler set to specific timezone

When a time zone is set to Scheduler through `timezone` property, the appointments will be displayed exactly based on the Scheduler timezone regardless of its client timezone. In the following code example, appointments will be displayed based on Eastern Time (UTC -05:00).

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject, }
from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 17)} eventSettings={eventSettings} timezone='America/New_York'>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    return (
        <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 17)} eventSettings={eventSettings} timezone='America/New_York'
        >
            <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
    )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Display events on same time everywhere with no time difference

Setting **timezone** to UTC for Scheduler will display the appointments on same time as in the database for all the users in different time zone.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { ScheduleComponent, Day, Week, Month, Timezone, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { fifaEventsData } from '../datasource';
const App = () => {
    const eventSettings = { dataSource: fifaEventsData };
    let timezone = new Timezone();
    const onCreate = () => {
        for (let fifaEvent of fifaEventsData) {
            let event = fifaEvent;
            event.StartTime = timezone.removeLocalOffset(event.StartTime);
            event.EndTime = timezone.removeLocalOffset(event.EndTime);
        }
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 5, 17)} created={onCreate} eventSettings={eventSettings}
timezone='UTC'>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, Month]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import {
    ScheduleComponent, Day, Week, Month, Timezone, Inject,
    ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { fifaEventsData } from '../datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: fifaEventsData };
    let timezone: Timezone = new Timezone();
    const onCreate = (): void => {
        for (let fifaEvent of fifaEventsData) {
            let event: { [key: string]: Object } = fifaEvent as { [key: string]:
Object };
            event.StartTime = timezone.removeLocalOffset(event.StartTime as Date);
            event.EndTime = timezone.removeLocalOffset(event.EndTime as Date);
        }
    }
    return (
        <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 5, 17)} created={onCreate} eventSettings={eventSettings}
timezone='UTC' >
            <ViewsDirective>
                <ViewDirective option='Day' />

```

```

        <ViewDirective option='Week' />
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, Month]} />
</ScheduleComponent>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>
<body>
    <div id='schedule'>

```

```

        <div id='loader'>Loading....</div>
      </div>
</body>
</html>

```

Set specific timezone for events

It is possible to set different timezone for Scheduler events by setting `startTimezone` and `endTimezone` properties within the `eventSettings` option. It allows each appointment to maintain different timezone and displays on Scheduler with appropriate time differences.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';
const App = () => {
  const scheduleData = [{
    Id: 3,
    Subject: 'Paris',
    StartTime: new Date(2018, 1, 15, 10, 0),
    EndTime: new Date(2018, 1, 15, 12, 30),
    StartTimezone: 'Europe/Moscow',
    EndTimezone: 'Europe/Moscow'
  }];
  const eventSettings = { dataSource: scheduleData };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
    Date(2018, 1, 11)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const scheduleData: Object[] = [{
    Id: 3,
    Subject: 'Paris',
    StartTime: new Date(2018, 1, 15, 10, 0),
    EndTime: new Date(2018, 1, 15, 12, 30),
    StartTimezone: 'Europe/Moscow',
    EndTimezone: 'Europe/Moscow'
  }];
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
    Date(2018, 1, 11)} eventSettings={eventSettings} >

```

```

    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008c8f;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

[Add or remove timezone names to/from the timezone collection](#)

Instead of displaying all the timezone names within the timezone collection (more than 200 are displayed on the editor window timezone fields by default), you can customize the timezone collection at application end as shown in the following example.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { useEffect } from 'react';
import { ScheduleComponent, Day, Week, Month, timezoneData, Inject } from
 '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const timeZones = [
        { Value: 'America/New_York', Text: '(UTC-05:00) Eastern Time' },
        { Value: 'UTC', Text: 'UTC' },
        { Value: 'Asia/Kolkata', Text: '(UTC+05:30) India Standard Time' }
    ];
    useEffect(() => {
        timezoneData.splice(0, timezoneData.length, timeZones);
    }, []);
    return (
        <ScheduleComponent width='100%' height='550px' selectedDate={new
        Date(2018, 1, 1)} eventSettings={eventSettings}>
            <Inject services={[Day, Week, Month]} />
        </ScheduleComponent>
    );
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { useEffect } from 'react';
import {
    ScheduleComponent, Day, Week, Month, timezoneData, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const timeZones: { [key: string]: Object }[] = [
        { Value: 'America/New_York', Text: '(UTC-05:00) Eastern Time' },
        { Value: 'UTC', Text: 'UTC' },
        { Value: 'Asia/Kolkata', Text: '(UTC+05:30) India Standard Time' }
    ];
    useEffect(() => {
        timezoneData.splice(0, timezoneData.length, timeZones as any);
    }, []);
    return (
```



```

    <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 1)} eventSettings={eventSettings} >
    <Inject services={[Day, Week, Month]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>

```

```
    </div>
  </body>
</html>
```

Timezone methods

offset

This method is used to calculate the difference between passed UTC date and timezone.

Parameters	Type	Description
----- ----- -----		
Date	Date	UTC time as date object.
Timezone	String	Timezone.

Returns **number**

```
`ts
// Assume your local timezone as IST/UTC+05:30
let timezone: Timezone = new Timezone();
let date: Date = new Date(2018,11,5,15,25,11);
let timeZoneOffset: number = timezone.offset(date,"Europe/Paris");
console.log(timeZoneOffset); //-60
`
```

convert

This method is used to convert the passed date from one timezone to another timezone.

Parameters	Type	Description
----- ----- -----		
Date	Date	UTC time as date object.
fromOffset	number/string	Timezone from which date need to be converted.
toOffset	number/string	Timezone to which date need to be converted.

Returns **Date**

```
`ts
// Assume your local timezone as IST/UTC+05:30
let timezone: Timezone = new Timezone();
let date: Date = new Date(2018,11,5,15,25,11);
let convertedDate: Date = timezone.convert(date, "Europe/Paris", "Asia/Tokya");
let convertedDate1: Date = timezone.convert(date, 60, -360);
console.log(convertedDate); //2018-12-05T08:55:11.000Z
console.log(convertedDate1); //2018-12-05T16:55:11.000Z
`
```

`

add

This method is used to add the time difference between passed UTC date and timezone.

Parameters	Type	Description
----- ----- -----		
Date	Date	UTC time as date object.
Timezone	String	Timezone.

Returns **Date**

`ts

```
// Assume your local timezone as IST/UTC+05:30
let timezone: Timezone = new Timezone();
let date: Date = new Date(2018,11,5,15,25,11);
let convertedDate: Date = timezone.add(date, "Europe/Paris");
console.log(convertedDate); //2018-12-05T05:25:11.000Z
```

`

remove

This method is used to remove the time difference between passed UTC date and timezone.

Parameters	Type	Description
----- ----- -----		
Date	Date	UTC as date object.
Timezone	String	Timezone.

Returns **Date**

`ts

```
// Assume your local timezone as IST/UTC+05:30
let timezone: Timezone = new Timezone();
let date: Date = new Date(2018,11,5,15,25,11);
let convertedDate: Date = timezone.remove(date, "Europe/Paris");
console.log(convertedDate); //2018-12-05T14:25:11.000Z
```

`

removeLocalOffset

This method is used to remove the local offset time from the date passed.

Parameters	Type	Description
----- ----- -----		
Date	Date	UTC as date object.

Returns `Date`

```
`ts
```

```
// Assume your local timezone as IST/UTC+05:30
```

```
let timezone: Timezone = new Timezone();
```

```
let date: Date = new Date(2018,11,5,15,25,11);
```

```
let convertedDate: Date = timezone.removeLocalOffset(date);
```

```
console.log(convertedDate); //2018-12-05T15:25:11.000Z
```

```
`
```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Views in React Schedule component

The Scheduler includes wide variety of view modes with unique configuration options for each view. The available view modes are listed below, out of which the [Link to the Video](#) view is set as active.

- Day
- Week
- Work week
- Month
- Year
- Agenda
- Month-Agenda
- Timeline Day
- Timeline Week
- Timeline Work Week
- Timeline Month
- Timeline Year

To navigate between different views and dates, the navigation options are available at the Scheduler header bar. The active view option is usually highlighted by default. The date range of the active view will also be displayed at the left corner of the header bar, clicking on which will open a calendar popup for the ease of desired date selection.

Learn how to customize each individual view of React Scheduler with different settings by watching this video:

By default, Scheduler displays the calendar views such as day, week, work week, month and agenda.

Setting specific view on scheduler

As the Scheduler displays `week` view by default, therefore to change the active view, set `currentView` property with the desired view name. The applicable view names that the Scheduler accepts are as follows,

- Day
- Week
- WorkWeek

- Month
- Year
- Agenda
- MonthAgenda
- TimelineDay
- TimelineWeek
- TimelineWorkWeek
- TimelineMonth
- TimelineYear

It is necessary to import and inject the appropriate view modules into the application to make use of these view modes on the Scheduler. Also, it is possible to display only the desired views on the Scheduler. To define and configure specific views, use the [views](#) property.

In the following example, the Scheduler displays 4 views namely, Week, Month, TimelineWeek and Timeline Month. The appropriate view modules are imported and injected properly to display those views on the Scheduler.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Week, Month, TimelineViews, TimelineMonth,
Inject, ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-
schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='Month' />
      <ViewDirective option='TimelineMonth' />
    </ViewsDirective>
    <Inject services={[Week, Month, TimelineViews, TimelineMonth]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Week, Month, TimelineViews, TimelineMonth, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData }
```

```

return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} >
  <ViewsDirective>
    <ViewDirective option='Week' />
    <ViewDirective option='TimelineWeek' />
    <ViewDirective option='Month' />
    <ViewDirective option='TimelineMonth' />
  </ViewsDirective>
  <Inject services={[Week, Month, TimelineViews, TimelineMonth]} />
</ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>

```

```

    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

ViewDirective and ViewsDirective needs to be imported to define views.

To configure Scheduler with simply 2 views, but with different configurations on each view, refer the following code example. Here, the Week view displays the dates in dd-MM-yyyy format whereas the Month view hides the weekend days and also displays it in readonly mode.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Week, Month, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Week' dateFormat='dd-MMM-yyyy' />
      <ViewDirective option='Month' showWeekend={false} readonly={true} />
    </ViewsDirective>
    <Inject services={[Week, Month]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, Month, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} >
    <ViewsDirective>
      <ViewDirective option='Week' dateFormat='dd-MMM-yyyy' />
      <ViewDirective option='Month' showWeekend={false} readonly={true} />
    </ViewsDirective>

```

```

    <Inject services={[Week, Month]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>

```


</html>

View specific configuration

There are scenarios where each view may need to have different configurations. For such cases, you can define the applicable scheduler properties within the `views` Property for each view option as depicted in the following examples. The fields available to be used within each view options are as follows.

Property	Type	Description	Applicable views
----------	------	-------------	------------------

option	View	It accepts the Scheduler view name, based on which we can define its related properties. The view names can be <code>Day</code> , <code>Week</code> and so on.	All views.
--------	------	--	------------

isSelected	Boolean	It acts similar to the <code>currentView</code> property and defines the active view of the Scheduler.	All views.
------------	---------	--	------------

dateFormat	Date	By default, Scheduler follows the date format as per the default culture assigned to it. When it is defined under specific view, only those assigned views follows this date format.	All views.
------------	------	--	------------

readonly	Boolean	When set to <code>true</code> , prevents the CRUD actions on the respective view under where it is defined.	All views.
----------	---------	---	------------

resourceHeaderTemplate	String	The template option which is used to customize the resource header cells on the Scheduler. It gets applied only on the views, wherever it is defined.	All views.
------------------------	--------	---	------------

dateHeaderTemplate	String	The template option which is used to customize the date header cells and is applied only on the views, wherever it is defined.	All views.
--------------------	--------	--	------------

eventTemplate	String	The template option to customize the events background. It will get applied to the events of the view to which it is currently being defined.	All views.
---------------	--------	---	------------

showWeekend	Boolean	When set to <code>false</code> , it hides the weekend days of a week from the views on which it is defined.	All views.
-------------	---------	---	------------

group	GroupModel	Allows to set different resource grouping options on all available Scheduler view modes.	All views.
-------	------------	--	------------

cellTemplate	String	The template option to customize the work cells of the Scheduler and is applied only on the views, on which it is defined.	Applicable on all views except Agenda view.
--------------	--------	--	---

workDays	Number[]	It is used to set the working days on the Scheduler views.	Applicable on all views except Agenda view.
----------	----------	--	---

displayName	String	When a particular view is customized to display with different intervals, this property allows the user to set different display name for each of the views.	Applicable on all views except Agenda and Month Agenda.
-------------	--------	--	---

interval	Number	It allows to customize the default Scheduler views with different set of days, weeks, work weeks or months on the applicable view type.	Applicable on all views except Agenda and Month Agenda.
----------	--------	---	---

startHour	String	It is used to specify the start hour, from which the Scheduler should be displayed. It accepts the time string in a short skeleton format and also, hides the time beyond the specified start	
-----------	--------	---	--

time. | Applicable on Day, Week, Work Week, Timeline Day, Timeline Week and Timeline Work Week views. |

| **endHour** | String | It is used to specify the end hour, at which the Scheduler ends. It accepts the time string in a short skeleton format. | Applicable on Day, Week, Work Week, Timeline Day, Timeline Week, and Timeline Work Week views. |

| **timeScale** | TimeScaleModel | Allows to set different timescale configuration on each applicable view modes. | Applicable on Day, Week, Work Week, Timeline Day, Timeline Week, and Timeline Work Week views. |

| **showWeekNumber** | Boolean | When set to **true**, shows the week number on the respective weeks. | Applicable on Day, Week, Work Week, and Month views. |

| **allowVirtualScrolling** | Boolean | It is used to enable or disable the virtual scrolling functionality. | Applicable on Agenda and Timeline views. |

| **headerRows** | HeaderRowsModel | Allows defining the custom header rows on timeline views of the Scheduler to display the year, month, week, date and hour label as an individual row. | Applicable only on all timeline views. |

Day view

Usually a day view displays a single day with all its related appointments. It is possible to customize the day view to display more number of days by extending the **ViewDirective** with **interval** option. You can also define any of the above defined properties within the **ViewDirective** definition as depicted in the following code example.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Inject, ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  const timeScale = { enable: true, slotCount: 5 };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' startHour='09:30' endHour='18:00'
timeScale={timeScale} />
    </ViewsDirective>
    <Inject services={[Day]} />
  </ScheduleComponent>;
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
```

```

ScheduleComponent, Day, Inject,
ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  const timeScale = { enable: true, slotCount: 5 };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' startHour='09:30' endHour='18:00'
timeScale={timeScale} />
    </ViewsDirective>
    <Inject services={[Day]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {

```

```

        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

All the above defined properties can be accessed within Day view except `allowVirtualScrolling` and `headerRows`.

Week view

The Week view displays a count of 7 days (from Sunday to Saturday) with all its related appointments. The first day of the week can be changed using the `firstDayOfWeek` which accepts the integer (Sunday=0, Monday=1, Tuesday=2 and so on) value. You can navigate to a particular date in day view from the week view by clicking on the appropriate dates on the date header bar.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, Inject, ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: appData };
    const timeScale = { enable: true, slotCount: 5 };
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' interval={2} displayName='2 Days' startHour='09:30' endHour='18:00' timeScale={timeScale} />
            <ViewDirective option='Week' interval={2} displayName='2 Weeks' showWeekend={false} isSelected={true} />
        </ViewsDirective>
        <Inject services={[Day, Week]} />
    </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {

```

```

ScheduleComponent, Day, Week, Inject,
ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  const timeScale = { enable: true, slotCount: 5 };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' interval={2} displayName='2 Days'
startHour='09:30' endHour='18:00' timeScale={timeScale} />
      <ViewDirective option='Week' interval={2} displayName='2 Weeks'
showWeekend={false} isSelected={true} />
    </ViewsDirective>
    <Inject services={[Day, Week]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>

```

```

<style>
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Work Week view

The Work week view displays only the working days of a week (count of 5 days) and its associated appointments. It is possible to customize the working days on the work week view by using the **workDays** property which accepts an array of integer values (such as Sunday=0, Monday=1, Tuesday=2 and so on). By default, it displays from Monday to Friday (5 days). You can also navigate to a particular date in the day view from the work week view by clicking on the appropriate dates in the date header bar.

The following code example depicts how to customize the resource header cells only on the **Work Week** view of the Scheduler.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, WorkWeek, Inject, ViewsDirective, ViewDirective }
from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  const workDays = [2, 3, 5];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='WorkWeek' workDays={workDays} />
    </ViewsDirective>
    <Inject services={[WorkWeek]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```
import {
  ScheduleComponent, WorkWeek, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  const workDays: number[] = [2, 3, 5];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='WorkWeek' workDays={workDays} />
    </ViewsDirective>
    <Inject services={[WorkWeek]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
```

```

        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

The Week, Work week and Day views can display the all-day row appointments in a separate all-day row with an expand/collapse option to view it.

Month view

A Month view displays the entire days of a particular month and all its related appointments. You can navigate to a particular date in the day view by clicking on the appropriate date text on the month cells.

By default, when you try to create an appointment through Month view, it is considered as created for an entire day. You can explicitly change this behavior by unchecking the **All-day** option from editor window, so that it defaults to the start time duration as 9.00 AM and end time as 9.30 AM.

You can also have the **+ more** text indicator on each day cell of a Month view, clicking on which will allows you to view the hidden appointments of a day.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Month, Inject, ViewsDirective, ViewDirective }
from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: appData };
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Month' showWeekNumber={true} readonly={true} />
        </ViewsDirective>
        <Inject services={[Month]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';

```



```
import {
  ScheduleComponent, Month, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Month' showWeekNumber={true} readonly={true} />
    </ViewsDirective>
    <Inject services={[Month]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
```

```

        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Year view

A Year view displays all the days of a particular year with months and all its related appointments. You can navigate to a particular date in the day view by clicking on the appropriate date text on the year cells.

Year view is available in both the **Horizontal** and **Vertical** orientations. You can manage the orientation of year view through **views** property.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Year, Inject, ViewsDirective, ViewDirective }
from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: appData };
    return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Year' showWeekNumber={true} readonly={true} />
        </ViewsDirective>
        <Inject services={[Year]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Year, Inject,
    ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: appData };

```

```

return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
  eventSettings={eventSettings}>
  <ViewsDirective>
    <ViewDirective option='Year' showWeekNumber={true} readonly={true} />
  </ViewsDirective>
  <Inject services={[Year]} />
</ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>

```

```

</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

The year view also has module support. In that, you can get all the months of a particular year in a calendar view format. In that calendar view, appointment contained dates are highlighted with dots placed under the individual date. When you click on the date, the event popup will be displayed and the events will be listed.

Agenda view

The Agenda view lists out the appointments in a grid-like view for the next 7 days by default from the current date. The count of the days can be changed using the API `agendaDaysCount`. It allows virtual scrolling of dates by enabling the `allowVirtualScrolling` property. Also, you can enable or disable the display of days on Scheduler that has no appointments by setting true or false to the `hideEmptyAgendaDays` property.

The following code example depicts how to customize the display of events within Agenda view alone.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Agenda, Inject, ViewsDirective, ViewDirective }
from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const eventSettings = { dataSource: appData };
  const instance = new Internationalization();
  const getTimeString = (value) => {
    return instance.formatDate(value, { skeleton: 'hm' });
  }
  const eventTemplate = (props) => {
    return (
      <div className="template-wrap">
        <div className="subject">{props.Subject}</div>
        <div className="time">
          Time: {getTimeString(props.StartTime)} -
          {getTimeString(props.EndTime)}</div>
        </div>
      </div>
    );
  }
  return (
    <ScheduleComponent width='100%' height='550px' agendaDaysCount={3}
      selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Agenda' eventTemplate={eventTemplate}
          allowVirtualScrolling={false} />
      </ViewsDirective>
      <Inject services={[Agenda]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));

```

```
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Agenda, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const eventSettings = { dataSource: appData };
  const instance: Internationalization = new Internationalization();
  const getTimeString = (value: Date) => {
    return instance.formatDate(value, { skeleton: 'hm' });
  }
  const eventTemplate = (props): JSX.Element => {
    return (
      <div className="template-wrap">
        <div className="subject">{props.Subject}</div>
        <div className="time">
          Time: {getTimeString(props.StartTime)} -
          {getTimeString(props.EndTime)}</div>
        </div>
      </div>
    );
  }
  return (
    <ScheduleComponent width='100%' height='550px' agendaDaysCount={3}
      selectedDate={new Date(2018, 1, 15)}
      eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Agenda' eventTemplate={eventTemplate}
          allowVirtualScrolling={false} />
      </ViewsDirective>
      <Inject services={[Agenda]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Schedule Height is mandatory to set in pixels for Agenda view alone.

Month Agenda view

A Month-Agenda view shows a month calendar, where clicking on a particular day will display the appointments present on that date below the calendar. The day with appointments are differentiated with a circular dot below the date of the calendar.

The following code example shows how to hide the weekend days on **MonthAgenda** view as well as the working days list is modified on Month Agenda view alone.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, MonthAgenda, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: appData };
    const workDays = [1, 2, 3];

```

```

    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 14)} eventSettings={eventSettings}>
    <ViewsDirective>
    <ViewDirective option='MonthAgenda' showWeekend={false}
workDays={workDays} />
    </ViewsDirective>
    <Inject services={[MonthAgenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, MonthAgenda, Inject,
    ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: appData };
    const workDays: number[] = [1, 2, 3];
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 14)}
    eventSettings={eventSettings}>
    <ViewsDirective>
    <ViewDirective option='MonthAgenda' showWeekend={false}
workDays={workDays} />
    </ViewsDirective>
    <Inject services={[MonthAgenda]} />
    </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Timeline views – Day, Week, Work Week

Similar to the day view, it shows a single day with all its appointments where the time slots are displayed horizontally. By default, the cell height adjusts as per the height set to Scheduler. When the number of appointments exceeds the visible area of the cells, the **+ more** text indicator will be displayed at the bottom to denote the presence of few more appointments in that time range.

To make use of the timeline views (Timeline Day, Timeline Week and Timeline Work Week) on Scheduler, import and inject the module **TimelineViews** from the **ej2-schedule** package.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, TimelineViews, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: appData };
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>

```



```

    <ViewDirective option='TimelineDay' startHour='10:00' endHour='15:30'
  />
  </ViewsDirective>
  <Inject services={[TimelineViews]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, TimelineViews, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' startHour='10:00' endHour='15:30'
    />
    </ViewsDirective>
    <Inject services={[TimelineViews]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Similar to the Week view, the timeline week view shows 7 days with its associated appointments with the time slots displayed horizontally.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, TimelineViews, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: appData };
    return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='TimelineWeek' timeScale={{ enable: false }} />
        </ViewsDirective>
        <Inject services={[TimelineViews]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, TimelineViews, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineWeek' timeScale={{ enable: false }} />
    </ViewsDirective>
    <Inject services={[TimelineViews]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
```

```

        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

The following code example depicts how to display the timeline work week view on Scheduler,

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, TimelineViews, Agenda, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: appData };
    const workDays = [2, 3, 5];
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='TimelineWorkWeek' interval={3}
workDays={workDays} dateFormat='dd-MMM-yyyy' />
            <ViewDirective option='Agenda' />
        </ViewsDirective>
        <Inject services={[TimelineViews, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, TimelineViews, Agenda, Inject,
    ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: appData };
    const workDays: number[] = [2, 3, 5];

```

```

return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>
        <ViewDirective option='TimelineWorkWeek' interval={3}
workDays={workDays} dateFormat='dd-MMM-yyyy' />
        <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[TimelineViews, Agenda]} />
    </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <link href="index.css" rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;

```

```

    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Clicking on the dates in the date header bar of Timeline day, Timeline week and Timeline work week will allow you to navigate to the Agenda view.

Timeline Month view

A Timeline Month view displays the current month days along with its appointments. To make use of the timeline Month view on Scheduler, import and inject `TimelineMonth` module from the `ej2-schedule` package.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, TimelineMonth, TimelineViews, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineMonth' showWeekend={false} />
      <ViewDirective option='TimelineDay' />
    </ViewsDirective>
    <Inject services={[TimelineMonth, TimelineViews]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, TimelineMonth, TimelineViews, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>

```

```

    <ViewDirective option='TimelineMonth' showWeekend={false} />
    <ViewDirective option='TimelineDay' />
  </ViewsDirective>
  <Inject services={[TimelineMonth, TimelineViews]} />
</ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>

```

```

        <div id='loader'>Loading....</div>
      </div>
</body>
</html>

```

Clicking on the dates in the date header bar of Timeline month will allow you to navigate to the Timeline day view.

Timeline Year view

In Timeline Year view, each row depicts a single resource. Whereas in the vertical view, each resource is grouped parallelly as columns. Here, the resource grouping follows the tree-view like hierarchical grouping structure and can contain any level of child resources.

To make use of the timeline Year view on Scheduler, import and inject `TimelineYear` module from the `ej2-schedule` package.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, TimelineYear, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineYear' displayName='Horizontal Timeline
Year' isSelected={true} />
    </ViewsDirective>
    <Inject services={[TimelineYear]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, TimelineYear, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineYear' displayName='Horizontal Timeline
Year' isSelected={true} />

```



```

    </ViewsDirective>
    <Inject services={[TimelineYear]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>

```

```
</body>
</html>
```

Resource grouping

The following code example depicts how to group the multiple resources on Timeline Year view and its relevant events are displayed accordingly under those resources.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, TimelineYear, ViewsDirective, ViewDirective,
ResourcesDirective, ResourceDirective, Inject } from '@syncfusion/ej2-react-
schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: resourceData };
  const group = { resources: ['Projects', 'Categories'] };
  const ownerData = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineYear' displayName='Horizontal Timeline
Year' isSelected={true} />
      <ViewDirective option='TimelineYear' displayName='Vertical Timeline
Year' orientation='Vertical' group={group} />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={ownerData} textField='OwnerText'
idField='Id' colorField='OwnerColor'>
      </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[TimelineYear]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
ScheduleComponent, TimelineYear, ViewsDirective, ViewDirective,
ResourcesDirective, ResourceDirective, Inject, EventSettingsModel,
GroupModel
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
```

```

const eventSettings: EventSettingsModel = { dataSource: resourceData };
const group: GroupModel = { resources: ['Projects', 'Categories'] };
const ownerData: Object[] = [
  { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
  { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
  { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
];
return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} eventSettings={eventSettings} >
  <ViewsDirective>
    <ViewDirective option='TimelineYear' displayName='Horizontal Timeline
Year' isSelected={true} />
    <ViewDirective option='TimelineYear' displayName='Vertical Timeline
Year' orientation='Vertical' group={group} />
  </ViewsDirective>
  <ResourcesDirective>
    <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true}
    dataSource={ownerData} textField='OwnerText' idField='Id'
colorField='OwnerColor'>
    </ResourceDirective>
  </ResourcesDirective>
  <Inject services={[TimelineYear]} />
</ScheduleComponent>)
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />

```

```

    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Auto row height

Timeline Year view supports Auto row height. When the feature `rowAutoHeight` is enabled, the row height gets auto-adjusted based on the number of overlapping events occupied in the same time range. If you disable the Auto row height, you have the + more text indicator on each day cell of a Timeline Year view, clicking on which will allow you to view the hidden appointments of a day.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, TimelineYear, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    return (<ScheduleComponent width='100%' height='550px' currentView='Month'
selectedDate={new Date(2018, 1, 17)} rowAutoHeight={true}
eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='TimelineYear' displayName='Horizontal Timeline
Year' isSelected={true} />
            <ViewDirective option='TimelineYear' displayName='Vertical Timeline
Year' orientation='Vertical' />
        </ViewsDirective>
        <Inject services={[TimelineYear]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, TimelineYear, Inject, ViewsDirective,
ViewDirective, EventSettingsModel } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return (<ScheduleComponent width='100%' height='550px' currentView='Month'
    selectedDate={new Date(2018, 1, 17)} rowAutoHeight={true}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='TimelineYear' displayName='Horizontal Timeline
Year' isSelected={true} />
      <ViewDirective option='TimelineYear' displayName='Vertical Timeline
Year' orientation='Vertical' />
    </ViewsDirective>
    <Inject services={[TimelineYear]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
```

```

<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Extending view intervals

It is possible to customize the display of default number of days on different Scheduler view modes. For example, a day view can be extended to display 3 days by setting the `interval` option as 3 for the `Day` option within the `ViewDirective` as depicted in the following code example. In the same way, you can also display 2 weeks by setting interval 2 for the `Week` option.

You can provide the alternative display name for such customized views on the Scheduler header bar, by setting the appropriate `displayName` property.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, Inject, ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: appData };
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' interval={3} displayName='3 Days' />

```

```

    <ViewDirective option='Week' interval={2} displayName='2 Weeks' />
  </ViewsDirective>
  <Inject services={[Day, Week]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { appData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: appData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' interval={3} displayName='3 Days' />
      <ViewDirective option='Week' interval={2} displayName='2 Weeks' />
    </ViewsDirective>
    <Inject services={[Day, Week]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

The view intervals can be extended on all the Scheduler view modes except Agenda and Month-Agenda views.

See Also

- [How to restrict view navigation while clicking on dates](#)

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Calendar mode in React Schedule component

The Scheduler supports the following two types of calendar mode.

- Gregorian Calendar
- Islamic Calendar

Gregorian Calendar

The Scheduler by default displays the gregorian calendar dates which is the most widely used calendar in the world. The Gregorian calendar is a solar calendar which has 12 months with 28 to 31 days each. A year which is divisible by four is said to be a leap year in this calendar mode. A leap year usually has 366 days whereas the regular year has 365 days.

Islamic Calendar

The Islamic Calendar, also known as the Hijri or Muslim calendar, is a lunar calendar which has 12 months in a year with 354 or 355 days. Each month of this calendar denotes the birth of the new lunar cycle and therefore, each month can have 29 or 30 days depending on the visibility of the moon. Here, the odd-numbered months have 30 days and the even months have 29 days.

The current Islamic year is 1440 AH. Usually the Gregorian calendar runs from approximately 11 September 2018 to 30 August 2019 for this 1440 AH year.

The Scheduler has a property `calendarMode` which is used to switch between the gregorian and islamic calendar modes. By default, it is set to `Gregorian` and to use it with Islamic calendar dates, define the `calendarMode` of Scheduler to `Islamic`. The following example depicts, how to display the Islamic calendar dates on Scheduler.

To make use of Islamic calendar in Scheduler, import the `Calendar` and `Islamic` module from `ej2-calendars` package and also inject it using the `Calendar.Inject` method. Apart from this, it requires the following CLDR data to be loaded using `loadCldr` function.

- `numberingSystems.json`
- `ca-gregorian.json`
- `numbers.json`
- `timeZoneNames.json`
- `ca-islamic.json`

To know more information on, how to install the CLDR data, refer the [Internationalization](#) topic.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, Month, TimelineViews, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
import { L10n, loadCldr } from '@syncfusion/ej2-base';
import * as localeObj from '../locale.json';
import * as numberingSystems from '../numberingSystems.json';
import * as gregorian from '../ca-gregorian.json';
import * as numbers from '../numbers.json';
import * as timeZoneNames from '../timeZoneNames.json';
import * as islamic from '../ca-islamic.json';
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames, islamic);
L10n.load(localeObj);
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    return (<ScheduleComponent height='550px' showQuickInfo={false}
selectedDate={new Date(2018, 1, 15)} locale='ar'
eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='TimelineWorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, Month, TimelineViews]} />
    </ScheduleComponent>
    </App>);
}
```

```

    </ScheduleComponent>);
  }
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, Month, TimelineViews, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
import { L10n, loadCldr } from '@syncfusion/ej2-base';
import * as localeObj from '../locale.json';
import * as numberingSystems from '../numberingSystems.json';
import * as gregorian from '../ca-gregorian.json';
import * as numbers from '../numbers.json';
import * as timeZoneNames from '../timeZoneNames.json';
import * as islamic from '../ca-islamic.json';
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames, islamic);
L10n.load(localeObj);
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (
    <ScheduleComponent height='550px' showQuickInfo={false} selectedDate={new
Date(2018, 1, 15)} locale='ar' eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='TimelineWorkWeek' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Day, Week, Month, TimelineViews]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Note: However, this feature does not yet support recurrence options, which we are planning to add them in the next release.

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) [Link to the Video](#) to know how to present and manipulate data.

Resources in React Schedule component

Resources and grouping support allows the Scheduler to be shared by multiple resources. Also, the appointments of each resource are displayed under relevant resources. Each resource in the Scheduler is arranged in a column/row wise order, with individual spacing to display all its respective appointments on a single page. It also supports the multiple levels of grouping of resources, thus enabling the categorization of resources in a hierarchical structure and shows it either in expandable groups (Timeline views) or else vertical hierarchy one after the other (Calendar views).

It is also possible to assign one or more resources to the same appointment, by allowing multiple selection of resource options available in the event editor window.

The HTML5 JavaScript Scheduler groups the resources based on different criteria. It includes grouping appointments based on resources, grouping resources based on dates, and timeline scheduling. Also, the data for resources bind with Scheduler either as a local JSON collection or URL, retrieving data from remote data services.

Learn how to add appointments of multiple resources to the React Scheduler from this video:

Resource fields

The default options available within the `resources` collection are as follows,

Field name	Type	Description
-----	-----	-----
<code>field</code>	String	A value that binds to the resource field of event object.
<code>title</code>	String	It holds the title of the resource field to be displayed on the event editor window.
<code>name</code>	String	A unique resource name used for differentiating various resource objects while grouping.
<code>allowMultiple</code>	Boolean	When set to <code>true</code> , allows multiple selection of resource names, thus creating multiple instances of same appointment for the selected resources.
<code>dataSource</code>	Object	Assigns the resource <code>dataSource</code> , where data can be passed either as an array of JavaScript objects, or else can create an instance of DataManager in case of processing remote data and can be assigned to the <code>dataSource</code> property. With the remote data assigned to <code>dataSource</code> , check the available adaptors to customize the data processing.
<code>query</code>	Query	Defines the external query that will be executed along with the data processing.
<code>idField</code>	String	Binds the resource ID field name from the resources <code>dataSource</code> .
<code>expandedField</code>	String	Binds the <code>expandedField</code> name from the resources <code>dataSource</code> . It usually holds boolean value which decide whether the resource of timeline views is in collapse or expand state on initial load.
<code>textField</code>	String	Binds the text field name from the resources <code>dataSource</code> . It usually holds the resource names.
<code>groupIDField</code>	String	Binds the group ID field name from the resource <code>dataSource</code> . It usually holds the value of resource IDs of parent level resources.
<code>colorField</code>	String	Binds the color field name from the resource <code>dataSource</code> . The color value mapped in this field will be applied to the events of resources.
<code>startHourField</code>	String	Binds the start hour field name from the resource <code>dataSource</code> . It allows to provide different work start hour for the resources.
<code>endHourField</code>	String	Binds the end hour field name from the resource <code>dataSource</code> . It allows to provide different work end hour for the resources.
<code>workDaysField</code>	String	Binds the work days field name from the resources <code>dataSource</code> . It allows to provide different working days collection for the resources.
<code>cssClassField</code>	String	Binds the custom CSS class field name from the resources <code>dataSource</code> . It maps the CSS class written for the specific resources and applies it to the events of those resources.

Resource data binding

The data for resources can bind with Scheduler either as a local JSON collection or a service URL, retrieving resource data from remote data services.

Using local JSON data

The following code example depicts how to bind the local JSON data to the `dataSource` of `resources` collection.

```
`ts
import * as React from 'react';
import { useState } from 'react';
import * as ReactDOM from 'react-dom';

import { Day, Week, WorkWeek, Month, Agenda, ScheduleComponent, ResourcesDirective,
ResourceDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';

const App = () => {
  const [ownerData] = useState([
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ]);

  const eventSettings: EventSettingsModel = { dataSource: resourceData };

  return (
    <ScheduleComponent width='100%' height='550px' selectedDate={new Date(2018, 3, 1)}
    eventSettings={eventSettings}>
      <ResourcesDirective>
        <ResourceDirective field='OwnerId' title='Owner' name='Owners' allowMultiple={true}
        dataSource={ownerData} textField='OwnerText' idField='Id' colorField='OwnerColor'>
        </ResourceDirective>
      </ResourcesDirective>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
  );
}

;

const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

,

[Using remote service URL](#)

The following code example depicts how to bind the remote data for resources `dataSource`.

```
`ts
import * as React from 'react';
import { useState } from 'react';
import * as ReactDOM from 'react-dom';

import { Week, Month, Agenda, ScheduleComponent, ResourcesDirective, ResourceDirective, Inject }
from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';

const App = () => {
  const [ownerData] = useState(new DataManager({
    url: 'Home/GetResourceData',
    adaptor: new UrlAdaptor(),
    crossDomain: true
  }));

  const eventSettings: EventSettingsModel = { dataSource: resourceData };

  return <ScheduleComponent width='100%' height='550px' selectedDate={new Date(2018, 3, 1)}
    eventSettings={eventSettings}>
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners' allowMultiple={true}
        dataSource={ownerData} textField='OwnerText' idField='Id' colorField='OwnerColor'>
      </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[Week, Month, Agenda]} />
  </ScheduleComponent>;
}

const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

,

[Scheduler with multiple resources](#)

It is possible to display the Scheduler in default mode without visually showcasing all the resources in it, but allowing to assign the required resources to the appointments through the event editor resource options.

The appointments belonging to the different resources will be displayed altogether on the default Scheduler, which will be differentiated based on the resource color assigned in the **resources** (depicting to which resource that particular appointment belongs) collection.

Example: To display default Scheduler with multiple resource options in the event editor, ignore the group option and simply define the **resources** property with all its internal options.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Week, Month, Agenda, ScheduleComponent, ViewsDirective,
ViewDirective, ResourcesDirective, ResourceDirective, Inject } from
'@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: resourceData }
  const ownerData = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={ownerData} textField='OwnerText'
idField='Id' colorField='OwnerColor'>
      </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[Week, Month, Agenda]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  Week, Month, Agenda, ScheduleComponent, ViewsDirective, ViewDirective,
  EventSettingsModel,
  ResourcesDirective, ResourceDirective, Inject
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: resourceData }
  const ownerData: Object[] = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
```

```

    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} eventSettings={eventSettings} >
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true}
      dataSource={ownerData} textField='OwnerText' idField='Id'
colorField='OwnerColor'>
    </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[Week, Month, Agenda]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>

```



```

<style>
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Setting `allowMultiple` to [Link to the Video](#) in the above code example allows you to select multiple resources from the event editor and also creates multiple copies of the same appointment in the Scheduler for each resources while rendering.

Resource grouping

Resource grouping support allows the Scheduler to group the resources in a hierarchical structure both as an expandable groups (Timeline views) and as vertical hierarchy displaying resources one after the other (Resources view).

Scheduler supports both single and multiple levels of resource grouping that can be customized both in timeline and vertical Scheduler views.

Explore the advanced options available with the multiple resources and grouping concepts of React Scheduler by watching this video:

Vertical resource view

The following code example displays how the multiple resources are grouped and its events are portrayed in the default calendar views.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Week, Month, Agenda, ScheduleComponent, ViewsDirective,
ViewDirective, ResourcesDirective, ResourceDirective, Inject } from
'@syncfusion/ej2-react-schedule';
import { timelineResourceData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: timelineResourceData }
  const group = { byGroupID: false, resources: ['Projects', 'Categories'] }
  const projectData = [
    { text: 'PROJECT 1', id: 1, color: '#cb6bb2' },
    { text: 'PROJECT 2', id: 2, color: '#56ca85' },
    { text: 'PROJECT 3', id: 3, color: '#df5286' },
  ];
  const categoryData = [
    { text: 'Development', id: 1, color: '#1aaa55' },
    { text: 'Testing', id: 2, color: '#7fa900' }
  ];

```

```

];
return (<ScheduleComponent width='100%' height='550px' currentView='Month'
selectedDate={new Date(2018, 3, 4)} eventSettings={eventSettings}
group={group}>
  <ViewsDirective>
    <ViewDirective option='Week' />
    <ViewDirective option='Month' />
    <ViewDirective option='Agenda' />
  </ViewsDirective>
  <ResourcesDirective>
    <ResourceDirective field='ProjectId' title='Choose Project'
name='Projects' allowMultiple={false} dataSource={projectData}
textField='text' idField='id' colorField='color'>
    </ResourceDirective>
    <ResourceDirective field='TaskId' title='Category' name='Categories'
allowMultiple={true} dataSource={categoryData} textField='text' idField='id'
colorField='color'>
    </ResourceDirective>
  </ResourcesDirective>
  <Inject services={[Week, Month, Agenda]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import {
  Week, Month, Agenda, ScheduleComponent, ViewsDirective,
  ViewDirective, EventSettingsModel, ResourcesDirective, ResourceDirective,
  Inject
} from '@syncfusion/ej2-react-schedule';
import { timelineResourceData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource:
timelineResourceData };
  const group = { byGroupID: false, resources: ['Projects', 'Categories'] };
  const projectData: Object[] = [
    { text: 'PROJECT 1', id: 1, color: '#cb6bb2' },
    { text: 'PROJECT 2', id: 2, color: '#56ca85' },
    { text: 'PROJECT 3', id: 3, color: '#df5286' },
  ];
  const categoryData: Object[] = [
    { text: 'Development', id: 1, color: '#1aaa55' },
    { text: 'Testing', id: 2, color: '#7fa900' }
  ];
  return (<ScheduleComponent width='100%' height='550px' currentView='Month'
selectedDate={new Date(2018, 3, 4)} eventSettings={eventSettings}
group={group} >
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>

```

```

    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='ProjectId' title='Choose Project'
name='Projects' allowMultiple={false}
      dataSource={projectData} textField='text' idField='id'
colorField='color'>
    </ResourceDirective>
      <ResourceDirective field='TaskId' title='Category' name='Categories'
allowMultiple={true}
      dataSource={categoryData} textField='text' idField='id'
colorField='color'>
    </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[Week, Month, Agenda]} />
  </ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;

```

```

        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Timeline resource view

The following code example depicts how to group the multiple resources on Timeline Scheduler views and its relevant events are displayed accordingly under those resources.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { TimelineViews, TimelineMonth, ScheduleComponent, ViewsDirective,
ViewDirective, ResourcesDirective, ResourceDirective, Inject } from
'@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: resourceData }
    const group = { resources: ['Rooms', 'Owners'] }
    const roomData = [
        { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
        { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
    ];
    const ownerData = [
        { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
        { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
        { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
    ];
    return (<ScheduleComponent width='100%' height='550px'
currentView='TimelineWeek' selectedDate={new Date(2018, 3, 1)}
eventSettings={eventSettings} group={group}>
        <ViewsDirective>
            <ViewDirective option='TimelineDay' />
            <ViewDirective option='TimelineWeek' />
            <ViewDirective option='TimelineMonth' />
        </ViewsDirective>
        <ResourcesDirective>
            <ResourceDirective field='RoomId' title='Room' name='Rooms'
dataSource={roomData} textField='RoomText' idField='Id'
colorField='RoomColor'>
            </ResourceDirective>
            <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={ownerData} textField='OwnerText'
idField='Id' groupIdField='GroupId' colorField='OwnerColor'>
            </ResourceDirective>
        </ResourcesDirective>
    </ScheduleComponent>

```

```

    <Inject services={[TimelineViews, TimelineMonth]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  TimelineViews, TimelineMonth, ScheduleComponent, ViewsDirective,
  ViewDirective,
  ResourcesDirective, ResourceDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: resourceData }
  const group = { resources: ['Rooms', 'Owners'] }
  const roomData: Object[] = [
    { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
    { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
  ]
  const ownerData: Object[] = [
    { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
  ];
  return (<ScheduleComponent width='100%' height='550px'
currentView='TimelineWeek' selectedDate={new Date(2018, 3, 1)}
eventSettings={eventSettings} group={group}>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='RoomId' title='Room' name='Rooms'
dataSource={roomData} textField='RoomText' idField='Id'
colorField='RoomColor'>
      </ResourceDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true}
dataSource={ownerData} textField='OwnerText' idField='Id'
groupIdField='GroupId' colorField='OwnerColor'>
      </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[TimelineViews, TimelineMonth]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Grouping single-level resources

This kind of grouping allows the Scheduler to display all the resources at a single level simultaneously. The appointments mapped under resources will be displayed with the colors as per the `colorField` defined on the resources collection.

Example: To display the Scheduler with single level resource grouping,

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Week, Month, TimelineViews, TimelineMonth, Agenda,
ScheduleComponent, ViewsDirective, ViewDirective, ResourcesDirective,
ResourceDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: resourceData }
  const group = { resources: ['Owners'] }
  const ownerData = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ];
  return (<ScheduleComponent width='100%' height='550px' currentView='Week'
selectedDate={new Date(2018, 3, 1)} eventSettings={eventSettings}
group={group}>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={ownerData} textField='OwnerText'
idField='Id' colorField='OwnerColor'>
      </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[Week, Month, TimelineViews, TimelineMonth, Agenda]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  Week, Month, TimelineViews, TimelineMonth, Agenda, ScheduleComponent,
  ViewsDirective, ViewDirective,
  ResourcesDirective, ResourceDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: resourceData }
  const group = { resources: ['Owners'] }
  const ownerData: Object[] = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ]

```

```

];
return (
  <ScheduleComponent width='100%' height='550px' currentView='Week'
selectedDate={new Date(2018, 3, 1)} eventSettings={eventSettings}
group={group}>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true}
      dataSource={ownerData} textField='OwnerText' idField='Id'
colorField='OwnerColor'>
      </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[Week, Month, TimelineViews, TimelineMonth, Agenda]}
/>
  </ScheduleComponent>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />

```



```

<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

The **name** field defined in the **resources** collection namely **Owners** will be mapped within the **group** property, in order to enable the grouping option with those resource levels on the Scheduler.

Grouping multi-level resources

It is possible to group the resources of Scheduler in multiple levels, by mapping the child resources to each parent resource. In the following example, there are 2 levels of resources, on which the second level resources are defined with **groupId** mapping to the first level resource's ID so as to establish the parent-child relationship between them.

Example: To display the Scheduler with multiple level resource grouping options,

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Week, Month, TimelineViews, TimelineMonth, ScheduleComponent,
ViewsDirective, ViewDirective, ResourcesDirective, ResourceDirective, Inject
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: resourceData }
  const group = { resources: ['Rooms', 'Owners'] }
  const roomData = [
    { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
    { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
  ];
  const ownerData = [
    { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
  ];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} eventSettings={eventSettings} group={group}>
    <ResourcesDirective>

```

```

    <ResourceDirective field='RoomId' title='Room' name='Rooms'
dataSource={roomData} textField='RoomText' idField='Id'
colorField='RoomColor'>
    </ResourceDirective>
    <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={ownerData} textField='OwnerText'
idField='Id' groupIDField='GroupId' colorField='OwnerColor'>
    </ResourceDirective>
</ResourcesDirective>
<ViewsDirective>
    <ViewDirective option='Week' />
    <ViewDirective option='Month' />
    <ViewDirective option='TimelineWeek' />
    <ViewDirective option='TimelineMonth' />
</ViewsDirective>
<Inject services={[Week, Month, TimelineViews, TimelineMonth]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    Week, Month, TimelineViews, TimelineMonth, Agenda, ScheduleComponent,
    ViewsDirective, ViewDirective,
    ResourcesDirective, ResourceDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: resourceData }
    const group = { resources: ['Rooms', 'Owners'] }
    const roomData: Object[] = [
        { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
        { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
    ]
    const ownerData: Object[] = [
        { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
        { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
        { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
    ];
    return (
        <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} eventSettings={eventSettings} group={group} >
            <ResourcesDirective>
                <ResourceDirective field='RoomId' title='Room' name='Rooms'
                    dataSource={roomData} textField='RoomText' idField='Id'
colorField='RoomColor'>
                </ResourceDirective>
                <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true}
                    dataSource={ownerData} textField='OwnerText' idField='Id'
groupIdField='GroupId' colorField='OwnerColor'>
            </ResourcesDirective>
        </ScheduleComponent>
    );
}

```

```

    </ResourceDirective>
  </ResourcesDirective>
  <ViewsDirective>
    <ViewDirective option='Week' />
    <ViewDirective option='Month' />
    <ViewDirective option='TimelineWeek' />
    <ViewDirective option='TimelineMonth' />
  </ViewsDirective>
  <Inject services={[Week, Month, TimelineViews, TimelineMonth]} />
</ScheduleComponent>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>

```

```

    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

One-to-One grouping

In multi-level grouping, Scheduler usually groups the resources on the child level based on the **GroupID** that maps with the **Id** field of parent level resources (as **byGroupID** set to true by default). There are also option which allows you to group all the child resource(s) against each of its parent resource(s). To enable this kind of grouping, set **false** to the **byGroupID** option within the **group** property. In the following code example, there are two levels of resources, on which all the 3 resources at the child level is mapped one to one with each resource on the first level.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Week, Month, TimelineViews, TimelineMonth, Agenda,
ScheduleComponent, ViewsDirective, ViewDirective, ResourcesDirective,
ResourceDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: resourceData }
  const group = { byGroupID: false, resources: ['Rooms', 'Owners'] }
  const roomData = [
    { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
    { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
  ];
  const ownerData = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' }
  ];
  return (<ScheduleComponent width='100%' height='550px' currentView='Week'
selectedDate={new Date(2018, 3, 1)} eventSettings={eventSettings}
group={group}>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='RoomId' title='Room' name='Rooms'
allowMultiple={false} dataSource={roomData} textField='RoomText' idField='Id'
colorField='RoomColor'>
      </ResourceDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={ownerData} textField='OwnerText'
idField='Id' colorField='OwnerColor'>

```

```

    </ResourceDirective>
  </ResourcesDirective>
  <Inject services={[Week, Month, TimelineViews, TimelineMonth, Agenda]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  Week, Month, TimelineViews, TimelineMonth, Agenda, ScheduleComponent,
  ViewsDirective, ViewDirective,
  ResourcesDirective, ResourceDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: resourceData }
  const group = { byGroupID: false, resources: ['Rooms', 'Owners'] }
  const roomData: Object[] = [
    { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
    { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
  ]
  const ownerData: Object[] = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' }
  ];
  return (<ScheduleComponent width='100%' height='550px' currentView='Week'
selectedDate={new Date(2018, 3, 1)} eventSettings={eventSettings}
group={group}>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='RoomId' title='Room' name='Rooms'
allowMultiple={false}
      dataSource={roomData} textField='RoomText' idField='Id'
colorField='RoomColor'>
      </ResourceDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true}
      dataSource={ownerData} textField='OwnerText' idField='Id'
colorField='OwnerColor'>
      </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[Week, Month, TimelineViews, TimelineMonth, Agenda]} />
  </ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));

```

```
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

Grouping resources by date

It groups the number of resources under each date and is applicable only on the calendar views such as Day, Week, Work Week, Month, Agenda and Month-Agenda. To enable such grouping, set `byDate` option to `true` within the `group` property.

Example: To display the Scheduler with resources grouped by date,

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Week, Month, Agenda, ScheduleComponent, ViewsDirective,
ViewDirective, ResourcesDirective, ResourceDirective, Inject } from
'@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: resourceData }
  const group = { byDate: true, resources: ['Owners'] }
  const resourceDatas = [
    { text: 'Alice', id: 1, color: '#1aaa55' },
    { text: 'Smith', id: 2, color: '#7fa900' }
  ];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} eventSettings={eventSettings} group={group}>
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={resourceDatas} textField='text' idField='id'
colorField='color'>
        </ResourceDirective>
      </ResourcesDirective>
      <ViewsDirective>
        <ViewDirective option='Week' />
        <ViewDirective option='Month' />
        <ViewDirective option='Agenda' />
      </ViewsDirective>
      <Inject services={[Week, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  Week, Month, Agenda, ScheduleComponent, ViewsDirective, ViewDirective,
  ResourcesDirective, ResourceDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: resourceData }
  const group = { byDate: true, resources: ['Owners'] }
  const resourceDatas: Object[] = [
    { text: 'Alice', id: 1, color: '#1aaa55' },
```

```

    { text: 'Smith', id: 2, color: '#7fa900' }
  ];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} eventSettings={eventSettings}
  group={group} >
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true}
        dataSource={resourceDatas} textField='text' idField='id'
colorField='color'>
      </ResourceDirective>
    </ResourcesDirective>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Week, Month, Agenda]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>

```



```

<style>
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

This kind of grouping by date is not applicable on any of the timeline views.

Working with shared events

Multiple resources can share the same events, thus allowing the CRUD action made on it to reflect on all other shared instances simultaneously. To enable such option, set `allowGroupEdit` option to `true` within the `group` property. With this property enabled, a single appointment

object will be maintained within the appointment collection, even if it is shared by more than one resource – whereas the resource fields of such appointment object will hold the IDs of the multiple resources.

Any actions such as create, edit or delete held on any one of the shared event instances, will be reflected on all other related instances visible on the UI.

Example: To edit all the resource events simultaneously,

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Week, Month, TimelineViews, TimelineMonth, Agenda,
ScheduleComponent, ResourcesDirective, ResourceDirective, ViewsDirective,
ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
const App = () => {
  const conferenceData = [
    { Text: 'Margaret', Id: 1, Color: '#1aaa55' },
    { Text: 'Robert', Id: 2, Color: '#357cd2' },
    { Text: 'Laura', Id: 3, Color: '#7fa900' }
  ];
  const data = [
    {
      Id: 1,
      Subject: 'Burning Man',
      StartTime: new Date(2018, 5, 1, 15, 0),
      EndTime: new Date(2018, 5, 1, 17, 0),
      ConferenceId: [1, 2, 3]
    }, {
      Id: 2,

```

```

        Subject: 'Data-Driven Economy',
        StartTime: new Date(2018, 5, 2, 12, 0),
        EndTime: new Date(2018, 5, 2, 14, 0),
        ConferenceId: [1, 2]
    }, {
        Id: 3,
        Subject: 'Techweek',
        StartTime: new Date(2018, 5, 2, 15, 0),
        EndTime: new Date(2018, 5, 2, 17, 0),
        ConferenceId: [2, 3]
    }, {
        Id: 4,
        Subject: 'Content Marketing World',
        StartTime: new Date(2018, 5, 2, 18, 0),
        EndTime: new Date(2018, 5, 2, 20, 0),
        ConferenceId: [1, 3]
    }, {
        Id: 5,
        Subject: 'B2B Marketing Forum',
        StartTime: new Date(2018, 5, 3, 10, 0),
        EndTime: new Date(2018, 5, 3, 12, 0),
        ConferenceId: [1, 2, 3]
    }
];
const eventSettings = { dataSource: data }
const group = { allowGroupEdit: true, resources: ['Conferences'] }
return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 5, 5)} currentView='WorkWeek' eventSettings={eventSettings}
group={group}>
    <ViewsDirective>
        <ViewDirective option='Week' />
        <ViewDirective option='Month' />
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='TimelineMonth' />
        <ViewDirective option='Agenda' />
    </ViewsDirective>
    <ResourcesDirective>
        <ResourceDirective field='ConferenceId' title='Conference'
name='Conferences' allowMultiple={true} dataSource={conferenceData}
textField='Text' idField='Id' colorField='Color'>
        </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[Week, Month, TimelineViews, TimelineMonth,
Agenda]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {

```

```

Week, Month, TimelineViews, TimelineMonth, EventSettingsModel, Agenda,
ScheduleComponent, ResourcesDirective, ResourceDirective, ViewsDirective,
ViewDirective, Inject
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const conferenceData: Object[] = [
    { Text: 'Margaret', Id: 1, Color: '#1aaa55' },
    { Text: 'Robert', Id: 2, Color: '#357cd2' },
    { Text: 'Laura', Id: 3, Color: '#7fa900' }
  ];
  const data: Object[] = [
    {
      Id: 1,
      Subject: 'Burning Man',
      StartTime: new Date(2018, 5, 1, 15, 0),
      EndTime: new Date(2018, 5, 1, 17, 0),
      ConferenceId: [1, 2, 3]
    }, {
      Id: 2,
      Subject: 'Data-Driven Economy',
      StartTime: new Date(2018, 5, 2, 12, 0),
      EndTime: new Date(2018, 5, 2, 14, 0),
      ConferenceId: [1, 2]
    }, {
      Id: 3,
      Subject: 'Techweek',
      StartTime: new Date(2018, 5, 2, 15, 0),
      EndTime: new Date(2018, 5, 2, 17, 0),
      ConferenceId: [2, 3]
    }, {
      Id: 4,
      Subject: 'Content Marketing World',
      StartTime: new Date(2018, 5, 2, 18, 0),
      EndTime: new Date(2018, 5, 2, 20, 0),
      ConferenceId: [1, 3]
    }, {
      Id: 5,
      Subject: 'B2B Marketing Forum',
      StartTime: new Date(2018, 5, 3, 10, 0),
      EndTime: new Date(2018, 5, 3, 12, 0),
      ConferenceId: [1, 2, 3]
    }
  ];
  const eventSettings: EventSettingsModel = { dataSource: data }
  const group = { allowGroupEdit: true, resources: ['Conferences'] }
  return (
    <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 5, 5)} currentView='WorkWeek' eventSettings={eventSettings}
group={group} >
      <ViewsDirective>
        <ViewDirective option='Week' />
        <ViewDirective option='Month' />
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='TimelineMonth' />
        <ViewDirective option='Agenda' />
      </ViewsDirective>
      <ResourcesDirective>

```

```

    <ResourceDirective field='ConferenceId' title='Conference'
name='Conferences' allowMultiple={true}
    dataSource={conferenceData} textField='Text' idField='Id'
colorField='Color'>
    </ResourceDirective>
</ResourcesDirective>
    <Inject services={[Week, Month, TimelineViews, TimelineMonth, Agenda]}
/>
    </ScheduleComponent>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>

```

```

    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Simple resource header customization

It is possible to customize the resource header cells using built-in template option and change the look and appearance of it in both the vertical and timeline view modes. All the resource related fields and other information can be accessed within the resource header template option.

Example: To customize the resource header and display it along with designation field, refer the below code example.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Week, Month, TimelineViews, TimelineMonth, Agenda,
ScheduleComponent, ViewsDirective, ViewDirective, ResourcesDirective,
ResourceDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { doctorData } from './datasource';
const App = () => {
    const getDoctorName = (value) => {
        return ((value.resourceData) ?
            value.resourceData[value.resource.textField] :
            value.resourceName);
    }
    const getDoctorLevel = (value) => {
        let resourceName = getDoctorName(value);
        return (resourceName === 'Will Smith') ? 'Cardiologist' : (resourceName
=== 'Alice') ? 'Neurologist' : 'Orthopedic Surgeon';
    }
    const eventSettings = { dataSource: doctorData };
    const group = { resources: ['Doctors'] };
    const resourceData = [
        { text: 'Will Smith', id: 1, color: '#ea7a57', designation: 'Cardioligst' },
        { text: 'Alice', id: 2, color: '#7fa900', designation: 'Neurologist' },
        { text: 'Robson', id: 3, color: '#7fa900', designation: 'Orthopedic
Surgeon' }
    ];
    const resourceHeaderTemplate = (props) => {
        return (<div className="template-wrap">
            <div className="resource-detail"><div className="resource-
name">{getDoctorName(props)}</div>
            <div className="resource-
designation">{getDoctorLevel(props)}</div></div></div>);
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} currentView='WorkWeek'
resourceHeaderTemplate={resourceHeaderTemplate} eventSettings={eventSettings}
group={group}>

```

```

    <ResourcesDirective>
      <ResourceDirective field='DoctorId' title='Doctor' name='Doctors'
dataSource={resourceData} textField='text' idField='id'
DesignationField='designation' colorField='color'>
    </ResourceDirective>
  </ResourcesDirective>
  <ViewsDirective>
    <ViewDirective option='Week' />
    <ViewDirective option='Month' />
    <ViewDirective option='TimelineWeek' />
    <ViewDirective option='TimelineMonth' />
    <ViewDirective option='Agenda' />
  </ViewsDirective>
  <Inject services={[Week, Month, TimelineViews, TimelineMonth, Agenda]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import {
  Week, Month, TimelineViews, TimelineMonth, Agenda, ScheduleComponent,
  GroupModel, ViewsDirective, EventSettingsModel, ViewDirective,
  ResourceDetails, ResourcesDirective, ResourceDirective, Inject, TreeViewArgs
} from '@syncfusion/ej2-react-schedule';
import { doctorData } from './datasource';
const App = () => {
  const getDoctorName = (value: ResourceDetails | TreeViewArgs): string => {
    return ((value as ResourceDetails).resourceData) ?
      (value as ResourceDetails).resourceData[(value as
ResourceDetails).resource.textField] :
      (value as TreeViewArgs).resourceName as string;
  }
  const getDoctorLevel = (value: ResourceDetails | TreeViewArgs): string => {
    let resourceName: string = getDoctorName(value);
    return (resourceName === 'Will Smith') ? 'Cardiologist' : (resourceName
=== 'Alice') ? 'Neurologist' : 'Orthopedic Surgeon';
  }
  const eventSettings: EventSettingsModel = { dataSource: doctorData };
  const group: GroupModel = { resources: ['Doctors'] };
  const resourceData: Object[] = [
    { text: 'Will Smith', id: 1, color: '#ea7a57', designation: 'Cardioligst'
},
    { text: 'Alice', id: 2, color: '#7fa900', designation: 'Neurologist' },
    { text: 'Robson', id: 3, color: '#7fa900', designation: 'Orthopedic
Surgeon' }
  ];
  const resourceHeaderTemplate = (props): JSX.Element => {
    return (<div className="template-wrap">
      <div className="resource-detail"><div className="resource-
name">{getDoctorName(props)}</div>

```

```

        <div className="resource-
designation">{getDoctorLevel(props)}</div></div></div>
    );
}
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} currentView='WorkWeek'
resourceHeaderTemplate={resourceHeaderTemplate} eventSettings={eventSettings}
group={group}>
    <ResourcesDirective>
        <ResourceDirective field='DoctorId' title='Doctor' name='Doctors'
dataSource={resourceData} textField='text' idField='id'
DesignationField='designation' colorField='color' >
        </ResourceDirective>
    </ResourcesDirective>
    <ViewsDirective>
        <ViewDirective option='Week' />
        <ViewDirective option='Month' />
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='TimelineMonth' />
        <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Week, Month, TimelineViews, TimelineMonth, Agenda]} />
</ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

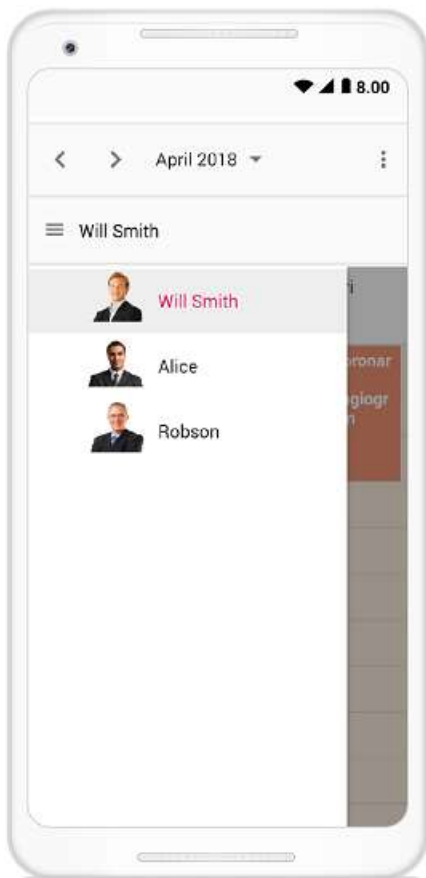
```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />

```

```
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

To customize the resource header in compact mode properly make use of the class



device as in the code example.

![Resource header template in compact mode](./images/header-template.png)

Customizing resource header with multiple columns

It is possible to customize the resource headers to display with multiple columns such as Room, Type and Capacity. The following code example depicts the way to achieve it and is applicable only on timeline views.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { TimelineViews, TimelineMonth, ScheduleComponent, ViewsDirective,
ViewDirective, ResourcesDirective, ResourceDirective, Inject } from
 '@syncfusion/ej2-react-schedule';
import { roomData } from './datasource';
const App = () => {
  const getRoomName = (value) => {
    return value.resourceData[value.resource.textField];
  }
  const getRoomType = (value) => {
    return value.resourceData.type;
  }
  const getRoomCapacity = (value) => {
    return value.resourceData.capacity;
  }
  const eventSettings = { dataSource: roomData };
  const group = { resources: ['MeetingRoom'] };
  const ownerData = [
    { text: 'Jammy', id: 1, color: '#ea7a57', capacity: 20, type:
'Conference' },
    { text: 'Tweety', id: 2, color: '#7fa900', capacity: 7, type: 'Cabin' },
    { text: 'Nestle', id: 3, color: '#5978ee', capacity: 5, type: 'Cabin' },
    { text: 'Phoenix', id: 4, color: '#fec200', capacity: 15, type:
'Conference' },
    { text: 'Mission', id: 5, color: '#df5286', capacity: 25, type:
'Conference' },
    { text: 'Hangout', id: 6, color: '#00bdae', capacity: 10, type: 'Cabin'
},
    { text: 'Rick Roll', id: 7, color: '#865fcf', capacity: 20, type:
'Conference' },
    { text: 'Rainbow', id: 8, color: '#1aaa55', capacity: 8, type: 'Cabin' },
    { text: 'Swarm', id: 9, color: '#df5286', capacity: 30, type:
'Conference' },
    { text: 'Photogenic', id: 10, color: '#710193', capacity: 25, type:
'Conference' }
  ];
  const resourceHeaderTemplate = (props) => {
    return (<div className="template-wrap">
      <div className="room-name">{getRoomName(props)}</div>
      <div className="room-type">{getRoomType(props)}</div>
      <div className="room-capacity">{getRoomCapacity(props)}</div>
    </div>);
  }
  const onRenderCell = (args) => {
    if (args.elementType === 'emptyCells' &&
args.element.classList.contains('e-resource-left-td')) {
      let target = args.element.querySelector('.e-resource-text');
```

```

        target.innerHTML = '<div class="name">Rooms</div><div
class="type">Type</div><div class="capacity">Capacity</div>';
    }
}
return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 7, 1)} currentView='TimelineWeek'
resourceHeaderTemplate={resourceHeaderTemplate} eventSettings={eventSettings}
renderCell={onRenderCell} group={group}>
    <ResourcesDirective>
        <ResourceDirective field='RoomId' title='Room Type' name='MeetingRoom'
dataSource={ownerData} textField='text' idField='id' colorField='color'>
        </ResourceDirective>
    </ResourcesDirective>
    <ViewsDirective>
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='TimelineMonth' />
    </ViewsDirective>
    <Inject services={[TimelineViews, TimelineMonth]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import {
    TimelineViews, TimelineMonth, ScheduleComponent, ViewsDirective,
    ViewDirective, RenderCellEventArgs,
    ResourceDetails, ResourcesDirective, ResourceDirective, Inject,
    EventSettingsModel, GroupModel
} from '@syncfusion/ej2-react-schedule';
import { roomData } from './datasource';
const App = () => {
    const getRoomName = (value: ResourceDetails) => {
        return (value as ResourceDetails).resourceData[(value as
ResourceDetails).resource.textField];
    }
    const getRoomType = (value: ResourceDetails) => {
        return (value as ResourceDetails).resourceData.type;
    }
    const getRoomCapacity = (value: ResourceDetails) => {
        return (value as ResourceDetails).resourceData.capacity;
    }
    const eventSettings: EventSettingsModel = { dataSource: roomData };
    const group: GroupModel = { resources: ['MeetingRoom'] };
    const ownerData: Object[] = [
        { text: 'Jammy', id: 1, color: '#ea7a57', capacity: 20, type:
'Conference' },
        { text: 'Tweety', id: 2, color: '#7fa900', capacity: 7, type: 'Cabin' },
        { text: 'Nestle', id: 3, color: '#5978ee', capacity: 5, type: 'Cabin' },
        { text: 'Phoenix', id: 4, color: '#fec200', capacity: 15, type:
'Conference' },

```

```

    { text: 'Mission', id: 5, color: '#df5286', capacity: 25, type:
'Conference' },
    { text: 'Hangout', id: 6, color: '#00bdae', capacity: 10, type: 'Cabin'
},
    { text: 'Rick Roll', id: 7, color: '#865fcf', capacity: 20, type:
'Conference' },
    { text: 'Rainbow', id: 8, color: '#1aaa55', capacity: 8, type: 'Cabin' },
    { text: 'Swarm', id: 9, color: '#df5286', capacity: 30, type:
'Conference' },
    { text: 'Photogenic', id: 10, color: '#710193', capacity: 25, type:
'Conference' }
  ];
  const resourceHeaderTemplate = (props): JSX.Element => {
    return (<div className="template-wrap">
      <div className="room-name">{getRoomName(props)}</div>
      <div className="room-type">{getRoomType(props)}</div>
      <div className="room-capacity">{getRoomCapacity(props)}</div>
    </div>
    );
  }
  const onRenderCell = (args: RenderCellEventArgs): void => {
    if (args.elementType === 'emptyCells' &&
args.element.classList.contains('e-resource-left-td')) {
      let target: HTMLElement = args.element.querySelector('.e-resource-
text') as HTMLElement;
      target.innerHTML = '<div class="name">Rooms</div><div
class="type">Type</div><div class="capacity">Capacity</div>';
    }
  }
  return (
    <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 7, 1)} currentView='TimelineWeek'
resourceHeaderTemplate={resourceHeaderTemplate} eventSettings={eventSettings}
renderCell={onRenderCell} group={group}>
      <ResourcesDirective>
        <ResourceDirective field='RoomId' title='Room Type'
name='MeetingRoom' dataSource={ownerData} textField='text' idField='id'
colorField='color' >
          </ResourceDirective>
        </ResourcesDirective>
        <ViewsDirective>
          <ViewDirective option='TimelineWeek' />
          <ViewDirective option='TimelineMonth' />
        </ViewsDirective>
        <Inject services={[TimelineViews, TimelineMonth]} />
      </ScheduleComponent>
    )
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>

```

```

<title>Syncfusion React Schedule</title>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta name="description" content="Essential JS 2 for React Components" />
<meta name="author" content="Syncfusion" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Collapse/Expand child resources in timeline views

It is possible to expand and collapse the resources which have child resource in timeline views dynamically. By default, resources are in expanded state with their child resource. We can collapse and expand the child resources in UI by setting `expandedField` option as `false` whereas its default value is `true`.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { TimelineViews, TimelineMonth, Agenda, ScheduleComponent,
ViewsDirective, ViewDirective, ResourcesDirective, ResourceDirective, Inject
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: resourceData }
  const group = { resources: ['Rooms', 'Owners'] }
  const roomData = [
    { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2', IsExpand: false },
    { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
  ];
  const ownerData = [
    { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
  ];
  return (<ScheduleComponent width='100%' height='550px'
currentView='TimelineWeek' selectedDate={new Date(2018, 3, 4)}
eventSettings={eventSettings} group={group}>
    <ViewsDirective>
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='RoomId' title='Room' name='Rooms'
dataSource={roomData} textField='RoomText' idField='Id'
colorField='RoomColor' expandedField='IsExpand'>
        </ResourceDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={ownerData} textField='OwnerText'
idField='Id' groupIDField='GroupId' colorField='OwnerColor'>
        </ResourceDirective>
      </ResourcesDirective>
    <Inject services={[TimelineViews, TimelineMonth, Agenda]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  TimelineViews, TimelineMonth, Agenda, ScheduleComponent, ViewsDirective,
  ViewDirective,
  ResourcesDirective, ResourceDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: resourceData }
  const group = { resources: ['Rooms', 'Owners'] }

```

```

const roomData: Object[] = [
  { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2', IsExpand: false },
  { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
]
const ownerData: Object[] = [
  { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
  { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
  { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
];
return (<ScheduleComponent width='100%' height='550px'
currentView='TimelineWeek' selectedDate={new Date(2018, 3, 4)}
eventSettings={eventSettings} group={group}>
  <ViewsDirective>
    <ViewDirective option='TimelineWeek' />
    <ViewDirective option='TimelineMonth' />
    <ViewDirective option='Agenda' />
  </ViewsDirective>
  <ResourcesDirective>
    <ResourceDirective field='RoomId' title='Room' name='Rooms'
      dataSource={roomData} textField='RoomText' idField='Id'
      colorField='RoomColor' expandedField='IsExpand'>
    </ResourceDirective>
    <ResourceDirective field='OwnerId' title='Owner' name='Owners'
      allowMultiple={true}
      dataSource={ownerData} textField='OwnerText' idField='Id'
      groupIDField='GroupId' colorField='OwnerColor'>
    </ResourceDirective>
  </ResourcesDirective>
  <Inject services={[TimelineViews, TimelineMonth, Agenda]} />
</ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Displaying tooltip for resource headers

It is possible to display tooltips over the resource headers showing the resource information. By default, there won't be any tooltips displayed on the resource headers, and to enable it, you need to assign the customized template design to the `headerTooltipTemplate` option within the `group` property.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Week, Month, TimelineViews, TimelineMonth, ScheduleComponent,
ViewsDirective, ViewDirective, ResourcesDirective, ResourceDirective, Inject
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: resourceData }
    const getRoomName = (value) => {
        return value.resourceData[value.resource.textField];
    }
    const headerTooltipTemplate = (props) => {
        return (<div className="template-wrap">
            <div className="room-name">{getRoomName(props)}</div>
        </div>);
    }
    const group = { resources: ['Rooms', 'Owners'], headerTooltipTemplate:
headerTooltipTemplate.bind(this) }
    const roomData = [
        { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },

```

```

    { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
  ];
  const ownerData = [
    { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
  ];
  return (<ScheduleComponent width='100%' height='550px'
currentView='TimelineWeek' selectedDate={new Date(2018, 3, 1)}
eventSettings={eventSettings} group={group}>
    <ResourcesDirective>
      <ResourceDirective field='RoomId' title='Room' name='Rooms'
dataSource={roomData} textField='RoomText' idField='Id'
colorField='RoomColor'>
        </ResourceDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={ownerData} textField='OwnerText'
idField='Id' groupIDField='GroupId' colorField='OwnerColor'>
        </ResourceDirective>
      </ResourcesDirective>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
    </ViewsDirective>
    <Inject services={[Week, Month, TimelineViews, TimelineMonth]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  Week, Month, TimelineViews, TimelineMonth, ScheduleComponent,
  ViewsDirective, ViewDirective,
  ResourcesDirective, ResourceDirective, Inject, ResourceDetails,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: resourceData }
  const getRoomName = (value: ResourceDetails) => {
    return (value as ResourceDetails).resourceData[(value as
ResourceDetails).resource.textField];
  }
  const headerTooltipTemplate = (props): JSX.Element => {
    return (<div className="template-wrap">
      <div className="room-name">{getRoomName(props)}</div>
    </div>
    );
  };
}

```



```

const group = { resources: ['Rooms', 'Owners'], headerTooltipTemplate:
headerTooltipTemplate.bind(this) }
const roomData: Object[] = [
  { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
  { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
]
const ownerData: Object[] = [
  { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
  { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
  { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
];
return (
  <ScheduleComponent width='100%' height='550px' currentView='TimelineWeek'
selectedDate={new Date(2018, 3, 1)} eventSettings={eventSettings}
group={group} >
    <ResourcesDirective>
      <ResourceDirective field='RoomId' title='Room' name='Rooms'
        dataSource={roomData} textField='RoomText' idField='Id'
        colorField='RoomColor'>
      </ResourceDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true}
        dataSource={ownerData} textField='OwnerText' idField='Id'
        groupIDField='GroupId' colorField='OwnerColor'>
      </ResourceDirective>
    </ResourcesDirective>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
    </ViewsDirective>
    <Inject services={[Week, Month, TimelineViews, TimelineMonth]} />
  </ScheduleComponent>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Choosing between resource colors for appointments

By default, the colors defined on the top level resources collection will be applied for the events. In case, if you want to apply specific resource color to events irrespective of its top-level parent resource color, it can be achieved by defining `resourceColorField` option within the `eventSettings` property.

In the following example, the colors mentioned in the second level will get applied over the events.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Week, Month, TimelineViews, TimelineMonth, Agenda,
ScheduleComponent, ViewsDirective, ViewDirective, ResourcesDirective,
ResourceDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import { resourceData } from './datasource';
const App = () => {
    const scheduleObj = useRef(null);
    const roomData = [
        { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },

```

```

    { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
  ];
  const ownerData = [
    { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
  ];
  let eventSettings = { dataSource: resourceData, resourceColorField:
  'Rooms' };
  const group = { resources: ['Rooms', 'Owners'] };
  const onChange = (args) => {
    scheduleObj.current.eventSettings.resourceColorField = args.value;
  }
  return (<div>
    <RadioButtonComponent value='Rooms' name='default' label='Rooms'
    checked={true} change={onChange}></RadioButtonComponent>
    <RadioButtonComponent value='Owners' name='default' label='Owners'
    checked={false} change={onChange}></RadioButtonComponent>
    <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
    currentView='Week' selectedDate={new Date(2018, 3, 1)}
    eventSettings={eventSettings} group={group}>
      <ViewsDirective>
        <ViewDirective option='Week' />
        <ViewDirective option='Month' />
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='TimelineMonth' />
        <ViewDirective option='Agenda' />
      </ViewsDirective>
      <ResourcesDirective>
        <ResourceDirective field='RoomId' title='Room' name='Rooms'
        dataSource={roomData} textField='RoomText' idField='Id' allowMultiple={false}
        colorField='RoomColor'>
        </ResourceDirective>
        <ResourceDirective field='OwnerId' title='Owner' name='Owners'
        allowMultiple={true} dataSource={ownerData} textField='OwnerText'
        idField='Id' groupIDField='GroupId' colorField='OwnerColor'>
        </ResourceDirective>
      </ResourcesDirective>
      <Inject services={[Week, Month, TimelineViews, TimelineMonth,
      Agenda]} />
    </ScheduleComponent>
  </div>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  Week, Month, TimelineViews, TimelineMonth, Agenda, ScheduleComponent,
  ViewsDirective, ViewDirective,

```

```

ResourcesDirective, ResourceDirective, Inject, EventSettingsModel,
GroupModel
} from '@syncfusion/ej2-react-schedule';
import { RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import { ChangeArgs } from '@syncfusion/ej2-buttons';
import { resourceData } from '../datasource';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const roomData: Object[] = [
    { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
    { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
  ]
  const ownerData: Object[] = [
    { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
  ];
  let eventSettings: EventSettingsModel = { dataSource: resourceData,
resourceColorField: 'Rooms' };
  const group: GroupModel = { resources: ['Rooms', 'Owners'] };
  const onChange = (args: ChangeArgs): void => {
    scheduleObj.current.eventSettings.resourceColorField = args.value;
  }
  return (
    <div>
      <RadioButtonComponent value='Rooms' name='default' label='Rooms'
checked={true} change={onChange} ></RadioButtonComponent>
      <RadioButtonComponent value='Owners' name='default' label='Owners'
checked={false} change={onChange} ></RadioButtonComponent>
      <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
currentView='Week' selectedDate={new Date(2018, 3, 1)}
eventSettings={eventSettings} group={group}>
        <ViewsDirective>
          <ViewDirective option='Week' />
          <ViewDirective option='Month' />
          <ViewDirective option='TimelineWeek' />
          <ViewDirective option='TimelineMonth' />
          <ViewDirective option='Agenda' />
        </ViewsDirective>
        <ResourcesDirective>
          <ResourceDirective field='RoomId' title='Room' name='Rooms'
            dataSource={roomData} textField='RoomText' idField='Id'
allowMultiple={false} colorField='RoomColor'>
          </ResourceDirective>
          <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true}
            dataSource={ownerData} textField='OwnerText' idField='Id'
groupIDField='GroupId' colorField='OwnerColor'>
          </ResourceDirective>
        </ResourcesDirective>
        <Inject services={[Week, Month, TimelineViews, TimelineMonth,
Agenda]} />
      </ScheduleComponent>
    </div>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));

```

```
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

The value of the `resourceColorField` field should be mapped with the `name` value given within the `resources` property.

Dynamically add and remove resources

It is possible to add or remove the resources dynamically to and from the Scheduler respectively. In the following example, when the checkboxes are checked and unchecked, the respective resources gets added up or removed from the Scheduler layout. To add new resource dynamically, `addResource` method is used which accepts the arguments such as resource object, resource name (within which level, the resource object to be added) and index (position where the resource needs to be added).

To remove the resources dynamically, `removeResource` method is used which accepts the index (position from where the resource to be removed) and resource name (within which level, the resource object presents) as parameters.

INDEX.JSX

```
import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { CheckBoxComponent } from '@syncfusion/ej2-react-buttons';
import { TimelineMonth, Month, ScheduleComponent, ViewsDirective,
ViewDirective, ResourcesDirective, ResourceDirective, Inject } from
 '@syncfusion/ej2-react-schedule';
import { holidayData, birthdayData, companyData, personalData } from
 './datasource';
import { Query, Predicate } from '@syncfusion/ej2-data';
const App = () => {
  const scheduleObj = useRef(null);
  const calendarCollections = [
    { CalendarText: 'My Calendar', CalendarId: 1, CalendarColor: '#c43081' },
    { CalendarText: 'Company', CalendarId: 2, CalendarColor: '#ff7f50' },
    { CalendarText: 'Birthday', CalendarId: 3, CalendarColor: '#AF27CD' },
    { CalendarText: 'Holiday', CalendarId: 4, CalendarColor: '#808000' }
  ];
  const generateCalendarData = () => {
    let collections = [];
    let dataCollections = [personalData, companyData, birthdayData,
holidayData];
    for (let data of dataCollections) {
      collections = collections.concat(data);
    }
    return collections;
  }
  let eventSettings = { dataSource: generateCalendarData() };
  const group = { resources: ['Calendars'] };
  const onChange = (args) => {
    const value =
parseInt((args.event.currentTarget).querySelector('input').getAttribute('value'), 10);
    const resourceData = calendarCollections.filter((item) => item.CalendarId
=== value);
    if (args.checked) {
      scheduleObj.current.addResource(resourceData[0], 'Calendars', value);
    }
    else {
      scheduleObj.current.removeResource(value, 'Calendars');
    }
  }
  return (<div>
```

```

<tbody>
  <tr style={{ height: '50px' }}>
    <td style={{ width: '100%' }}>
      <CheckBoxComponent value='1' id='personal' checked={true} label='My
Calendar' disabled={true} change={onChange}></CheckBoxComponent>
      <CheckBoxComponent value='2' id='company' checked={false}
label='Company' change={onChange}></CheckBoxComponent>
      <CheckBoxComponent value='3' id='birthdays' checked={false}
label='Birthday' change={onChange}></CheckBoxComponent>
      <CheckBoxComponent value='4' id='holidays' checked={false}
label='Holiday' change={onChange}></CheckBoxComponent>
    </td>
  </tr>
</tbody>
<ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate={new Date(2018, 3, 1)} group={group}
eventSettings={eventSettings}>
  <ResourcesDirective>
    <ResourceDirective field='CalendarId' title='Calendar'
name='Calendars' allowMultiple={true} dataSource={[calendarCollections[0]]}
textField='CalendarText' idField='CalendarId' colorField='CalendarColor'>
    </ResourceDirective>
  </ResourcesDirective>
  <ViewsDirective>
    <ViewDirective option='Month' />
    <ViewDirective option='TimelineMonth' />
  </ViewsDirective>
  <Inject services={[TimelineMonth, Month]} />
</ScheduleComponent>
</div>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { CheckBoxComponent, ChangeEventArgs } from '@syncfusion/ej2-react-
buttons';
import {
  TimelineMonth, Month, ScheduleComponent,
  ViewsDirective, ViewDirective, ResourcesDirective, ResourceDirective,
  Inject, EventSettingsModel, GroupModel
} from '@syncfusion/ej2-react-schedule';
import { holidayData, birthdayData, companyData, personalData } from
'./datasource';
import { Predicate, Query } from '@syncfusion/ej2-data';
import { CheckBox } from '@syncfusion/ej2-buttons';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const calendarCollections: Object[] = [
    { CalendarText: 'My Calendar', CalendarId: 1, CalendarColor: '#c43081' },
    { CalendarText: 'Company', CalendarId: 2, CalendarColor: '#ff7f50' },

```

```

    { CalendarText: 'Birthday', CalendarId: 3, CalendarColor: '#AF27CD' },
    { CalendarText: 'Holiday', CalendarId: 4, CalendarColor: '#808000' }
  ];
  const generateCalendarData = (): Object[] => {
    let collections: Object[] = [];
    let dataCollections: Object[][] = [personalData, companyData,
    birthdayData, holidayData];
    for (let data of dataCollections) {
      collections = collections.concat(data);
    }
    return collections;
  }
  let eventSettings: EventSettingsModel = { dataSource:
  generateCalendarData() };
  const group: GroupModel = { resources: ['Calendars'] };
  const onChange = (args: ChangeEventArgs): void => {
    const value =
    parseInt((args.event.currentTarget).querySelector('input').getAttribute('value'), 10);
    const resourceData = calendarCollections.filter((item: any) =>
    item.CalendarId === value);
    if(args.checked) {
      scheduleObj.current.addResource(resourceData[0], 'Calendars', value);
    }
    else {
      scheduleObj.current.removeResource(value, 'Calendars');
    }
  }
  return (
    <div>
      <tbody>
        <tr style={{ height: '50px' }}>
          <td style={{ width: '100%' }}>
            <CheckBoxComponent value='1' id='personal' checked={true} label='My
            Calendar' disabled={true} change={onChange}></CheckBoxComponent>
            <CheckBoxComponent value='2' id='company' checked={false}
            label='Company' change={onChange}></CheckBoxComponent>
            <CheckBoxComponent value='3' id='birthdays' checked={false}
            label='Birthday' change={onChange}></CheckBoxComponent>
            <CheckBoxComponent value='4' id='holidays' checked={false}
            label='Holiday' change={onChange}></CheckBoxComponent>
          </td>
        </tr>
      </tbody>
      <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
      selectedDate={new Date(2018, 3, 1)} group={group}
      eventSettings={eventSettings} >
        <ResourcesDirective>
          <ResourceDirective field='CalendarId' title='Calendar'
          name='Calendars' allowMultiple={true}
          dataSource={[calendarCollections[0]]} textField='CalendarText'
          idField='CalendarId' colorField='CalendarColor'>
            </ResourceDirective>
          </ResourcesDirective>
        < ViewsDirective >
          <ViewDirective option='Month' />
          <ViewDirective option='TimelineMonth' />
        </ViewsDirective>
      </ScheduleComponent>
    </div>
  );

```



```

        </ViewsDirective>
        <Inject services={[TimelineMonth, Month]} />
    </ScheduleComponent>
</div>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>

```

```

    </div>
  </body>
</html>

```

Setting different working days and hours for resources

Each resource in the Scheduler can have different working hours as well as different working days set to it. There are default options available within the `resources` collection, to customize the default working hours and days of the Scheduler.

Set different work days

Different working days can be set for the resources of Scheduler using the `workDaysField` property which maps the working days field from the resource dataSource. This field accepts the collection of day indexes (from 0 to 6) of a week. By default, it is set to [1, 2, 3, 4, 5] and in the following example, each resource has been set with different values and therefore each of them will render only those working days. This option is applicable only on the calendar views and is not applicable on timeline views.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { WorkWeek, Month, TimelineViews, ScheduleComponent, ViewsDirective,
ViewDirective, ResourcesDirective, ResourceDirective, Inject } from
'@syncfusion/ej2-react-schedule';
import { doctorData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: doctorData }
  const group = { resources: ['Doctors'] }
  const resourceData = [
    { text: 'Will Smith', id: 1, color: '#ea7a57', workDays: [1, 2, 4, 5] },
    { text: 'Alice', id: 2, color: '#357cd2', workDays: [1, 3, 5] },
    { text: 'Robson', id: 3, color: '#7fa900', workDays: [2, 6] }
  ];
  return (
    <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} currentView='WorkWeek' eventSettings={eventSettings}
group={group}>
      <ResourcesDirective>
        <ResourceDirective field='DoctorId' title='Doctor' name='Doctors'
dataSource={resourceData} textField='text' idField='id' colorField='color'
workDaysField='workDays'>
        </ResourceDirective>
      </ResourcesDirective>
      <ViewsDirective>
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[TimelineViews, WorkWeek, Month]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import {
  WorkWeek, Month, TimelineViews, ScheduleComponent, ViewsDirective,
  ViewDirective,
  ResourcesDirective, ResourceDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { doctorData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: doctorData }
  const group = { resources: ['Doctors'] }
  const resourceData: Object[] = [
    { text: 'Will Smith', id: 1, color: '#ea7a57', workDays: [1, 2, 4, 5] },
    { text: 'Alice', id: 2, color: '#357cd2', workDays: [1, 3, 5] },
    { text: 'Robson', id: 3, color: '#7fa900', workDays: [2, 6] }
  ];
  return (
    <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 3, 1)} currentView='WorkWeek' eventSettings={eventSettings}
group={group}>
      <ResourcesDirective>
        <ResourceDirective field='DoctorId' title='Doctor' name='Doctors'
dataSource={resourceData} textField='text' idField='id' colorField='color'
workDaysField='workDays' >
          </ResourceDirective>
        </ResourcesDirective>
      <ViewsDirective>
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[TimelineViews, WorkWeek, Month]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Set different work hours

Working hours indicates the work hour duration of a day, which is highlighted visually with active color over the work cells. Each resource on the Scheduler can be defined with its own set of working hours as depicted in the following example.

- **startHourField** - Denotes the start time of the working/business hour in a day.
- **endHourField** - Denotes the end time limit of the working/business hour in a day.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import { WorkWeek, Month, TimelineViews, ScheduleComponent, ViewsDirective,
ViewDirective, ResourcesDirective, ResourceDirective, Inject } from
'@syncfusion/ej2-react-schedule';
import { doctorData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: doctorData }
    const group = { resources: ['Doctors'] }
    const resourceData = [

```

```

    { text: 'Will Smith', id: 1, color: '#ea7a57', workDays: [1, 2, 4, 5],
    startHour: '07:00', endHour: '13:00' },
    { text: 'Alice', id: 2, color: '#357cd2', workDays: [1, 3, 5], startHour:
    '09:00', endHour: '17:00' },
    { text: 'Robson', id: 3, color: '#7fa900', startHour: '08:00', endHour:
    '16:00' }
  ];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
  Date(2018, 3, 1)} currentView='WorkWeek' eventSettings={eventSettings}
  group={group}>
    <ResourcesDirective>
      <ResourceDirective field='DoctorId' title='Doctor' name='Doctors'
      dataSource={resourceData} textField='text' idField='id' colorField='color'
      workDaysField='workDays' startHourField='startHour' endHourField='endHour'>
        </ResourceDirective>
      </ResourcesDirective>
    <ViewsDirective>
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[TimelineViews, WorkWeek, Month]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from "react-dom";
import {
  WorkWeek, Month, TimelineViews, ScheduleComponent, ViewsDirective,
  ViewDirective,
  ResourcesDirective, ResourceDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { doctorData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: doctorData }
  const group = { resources: ['Doctors'] }
  const resourceData: Object[] = [
    { text: 'Will Smith', id: 1, color: '#ea7a57', workDays: [1, 2, 4, 5],
    startHour: '07:00', endHour: '13:00' },
    { text: 'Alice', id: 2, color: '#357cd2', workDays: [1, 3, 5], startHour:
    '09:00', endHour: '17:00' },
    { text: 'Robson', id: 3, color: '#7fa900', startHour: '08:00', endHour:
    '16:00' }
  ];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
  Date(2018, 3, 1)} currentView='WorkWeek' eventSettings={eventSettings}
  group={group}>
    <ResourcesDirective>
      <ResourceDirective field='DoctorId' title='Doctor' name='Doctors'
      dataSource={resourceData} textField='text' idField='id' colorField='color'

```

```

        workDaysField='workDays' startHourField='startHour'
endHourField='endHour' >
    </ResourceDirective>
</ResourcesDirective>
<ViewsDirective>
    <ViewDirective option='TimelineWeek' />
    <ViewDirective option='WorkWeek' />
    <ViewDirective option='Month' />
</ViewsDirective>
<Inject services={[TimelineViews, WorkWeek, Month]} />
</ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;

```

```

    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

In this example, a resource named **Will Smith** is depicted with working hours ranging from 7.00 AM to 3.00 PM and is visually illustrated with active colors, whereas the other two resources have different working hours set.

Hide non-working days when grouped by date

In Scheduler, you can set custom work days for each resource and group the Scheduler by date to display these work days. By default, the Scheduler will show all days when it is grouped by date, even if they are not included in the custom work days for the resources. However, you can use the [hideNonWorkingDays](#) property to only display the custom work days in the Scheduler.

To use the [hideNonWorkingDays](#) property, you need to include it in the configuration options for your Scheduler component. Set the value of [hideNonWorkingDays](#) to **true** to enable this feature.

Example: To display the Scheduler with resources grouped by date for custom working days,

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Week, Month, Agenda, ScheduleComponent, ViewsDirective,
ViewDirective, ResourcesDirective, ResourceDirective, Inject } from
'@syncfusion/ej2-react-schedule';
const App = () => {
  const group = { byDate: true, resources: ['Owners'], hideNonWorkingDays:
true }
  const resourceData = [
    { text: 'Alice', id: 1, color: '#1aaa55', workDays: [1, 2, 3, 4] },
    { text: 'Smith', id: 2, color: '#7fa900', workDays: [2, 3, 5] }
  ];
  return (<ScheduleComponent width='100%' height='550px' group={group}>
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={resourceData} textField='text' idField='id'
colorField='color' workDaysField='workDays'>
    </ResourceDirective>
    </ResourcesDirective>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Week, Month, Agenda]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));

```

```
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  Week, Month, Agenda, ScheduleComponent, ViewsDirective, ViewDirective,
  ResourcesDirective, ResourceDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const group = { byDate: true, resources: ['Owners'], hideNonWorkingDays:
true }
  const resourceData: Object[] = [
    { text: 'Alice', id: 1, color: '#1aaa55', workDays: [1, 2, 3, 4] },
    { text: 'Smith', id: 2, color: '#7fa900', workDays: [2, 3, 5] }
  ];
  return (<ScheduleComponent width='100%' height='550px'
    group={group} >
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true}
        dataSource={resourceData} textField='text' idField='id'
colorField='color' workDaysField='workDays'>
      </ResourceDirective>
    </ResourcesDirective>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Week, Month, Agenda]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
```



```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

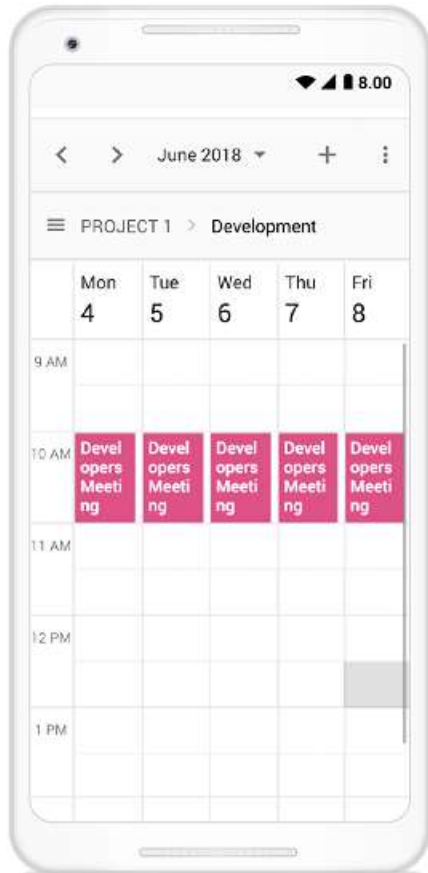
```

The [hideNonWorkingDays](#) property only applies when the Scheduler is grouped [byDate](#).

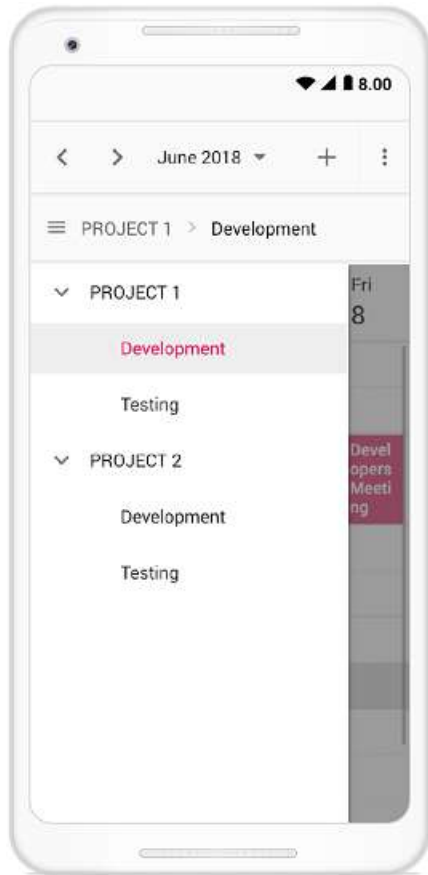
Compact view in mobile

Although the Scheduler views are designed keeping in mind the responsiveness of the control in mobile devices, however when using Scheduler with multiple resources - it is difficult to view all the resources and its relevant events at once on the mobile. Therefore, we have introduced a new compact mode specially for displaying multiple resources of Scheduler on mobile devices. By default, this mode is enabled while using Scheduler with multiple resources on mobile devices. If in case, you need to disable this compact mode, set `false` to the `enableCompactView` option within the `group` property. Disabling this option will display the exact desktop mode of Scheduler view on mobile devices.

With this compact view enabled on mobile, you can view only single resource at a time and to switch to other resources, there is a tree view at the left listing out all other available resources - clicking on which will display that particular resource and its related appointments.



Clicking on the menu icon before the resource text will show the resources available in the Scheduler as following.



Adaptive UI in desktop

By default, the Scheduler layout adapts automatically in the desktop and mobile devices with appropriate UI changes. In case, if the user wants to display the Adaptive scheduler in desktop mode with adaptive enhancements, then the property `enableAdaptiveUI` can be set to true. Enabling this option will display the exact mobile mode of Scheduler view on desktop devices.

Some of the default changes made for compact Scheduler to render in desktop devices are as follows,

- View options displayed in the Navigation drawer.
- Plus icon is added to the header for new event creation.
- Today icon is added to the header instead of the Today button.
- With Multiple resources – only one resource has been shown to enhance the view experience of resource events details clearly. To switch to other resources, there is a TreeView on the left that lists all other available resources, clicking on which will display that particular resource and its related events.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, Month, Year, Resize, DragAndDrop,
Inject, ResourcesDirective, ResourceDirective, ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { resourceData, timelineResourceData } from '../datasource';
const App = () => {
```

```

const eventSettings = { dataSource:
resourceData.concat(timelineResourceData) }
const projectData = [
  { text: 'PROJECT 1', id: 1, color: '#cb6bb2' },
  { text: 'PROJECT 2', id: 2, color: '#56ca85' },
  { text: 'PROJECT 3', id: 3, color: '#df5286' }
];
const categoryData = [
  { text: 'Nancy', id: 1, groupId: 1, color: '#df5286' },
  { text: 'Steven', id: 2, groupId: 1, color: '#7fa900' },
  { text: 'Robert', id: 3, groupId: 2, color: '#ea7a57' },
  { text: 'Smith', id: 4, groupId: 2, color: '#5978ee' },
  { text: 'Micheal', id: 5, groupId: 3, color: '#df5286' },
  { text: 'Root', id: 6, groupId: 3, color: '#00bdae' }
];
const group = { resources: ['Projects', 'Categories'] };
return (<ScheduleComponent width='100%' height='650px' id='schedule'
selectedDate={new Date(2018, 3, 4)} group={group} enableAdaptiveUI={true}
currentView='Month' eventSettings={eventSettings}>
  <ViewsDirective>
    <ViewDirective option='Day' />
    <ViewDirective option='Week' />
    <ViewDirective option='Month' />
  </ViewsDirective>
  <ResourcesDirective>
    <ResourceDirective field='ProjectId' title='Choose Project'
name='Projects' allowMultiple={false} dataSource={projectData}
textField='text' idField='id' colorField='color'>
    </ResourceDirective>
    <ResourceDirective field='TaskId' title='Category' name='Categories'
allowMultiple={true} dataSource={categoryData} textField='text' idField='id'
groupIdField='groupId' colorField='color'>
    </ResourceDirective>
  </ResourcesDirective>
  <Inject services={[Day, Week, Month, Year, Resize, DragAndDrop]} />
</ScheduleComponent>);
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, Month, Year, Resize, EventSettingsModel,
  DragAndDrop, Inject, ResourcesDirective, ResourceDirective, GroupModel,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { resourceData, timelineResourceData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource:
resourceData.concat(timelineResourceData) }
  const projectData: Object[] = [
    { text: 'PROJECT 1', id: 1, color: '#cb6bb2' },
    { text: 'PROJECT 2', id: 2, color: '#56ca85' },

```

```

    { text: 'PROJECT 3', id: 3, color: '#df5286' }
  ];
  const categoryData: Object[] = [
    { text: 'Nancy', id: 1, groupId: 1, color: '#df5286' },
    { text: 'Steven', id: 2, groupId: 1, color: '#7fa900' },
    { text: 'Robert', id: 3, groupId: 2, color: '#ea7a57' },
    { text: 'Smith', id: 4, groupId: 2, color: '#5978ee' },
    { text: 'Micheal', id: 5, groupId: 3, color: '#df5286' },
    { text: 'Root', id: 6, groupId: 3, color: '#00bdae' }
  ];
  const group: GroupModel = { resources: ['Projects', 'Categories'] };
  return (
    <ScheduleComponent width='100%' height='650px' id='schedule'
      selectedDate={new Date(2018, 3, 4)} group={group}
      enableAdaptiveUI={true} currentView='Month' eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <ResourcesDirective>
        <ResourceDirective field='ProjectId' title='Choose Project'
          name='Projects' allowMultiple={false}
          dataSource={projectData} textField='text' idField='id'
          colorField='color'>
        </ResourceDirective>
        <ResourceDirective field='TaskId' title='Category' name='Categories'
          allowMultiple={true}
          dataSource={categoryData} textField='text' idField='id'
          groupIdField='groupId' colorField='color'>
        </ResourceDirective>
      </ResourcesDirective>
      <Inject services={[Day, Week, Month, Year, Resize, DragAndDrop]} />
    </ScheduleComponent>
  );
}
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Header rows in React Schedule component

The Timeline views can have additional header rows other than its default date and time header rows. It is possible to show individual header rows for displaying year, month and week separately using the `HeaderRowDirective`. This is applicable only on the timeline views. The possible rows which can be added using `HeaderRowDirective` are as follows.

- Year
- Month
- Week
- Date
- Hour

The [Link to the Video](#) row is not applicable for Timeline month view.

Learn to add and customize additional header rows in the Timeline views of React Scheduler from this video:

The following example shows the Scheduler displaying all the available header rows on timeline views.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, HeaderRowDirective, HeaderRowsDirective,
TimelineViews, Inject, ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 11, 31)} eventSettings={eventSettings} startHour='09:00'
endHour='13:00'>
    <HeaderRowsDirective>
      <HeaderRowDirective option='Year' />
      <HeaderRowDirective option='Month' />
      <HeaderRowDirective option='Week' />
      <HeaderRowDirective option='Date' />
      <HeaderRowDirective option='Hour' />
    </HeaderRowsDirective>
    <ViewsDirective>
      <ViewDirective option='TimelineWeek' />
    </ViewsDirective>
    <Inject services={[TimelineViews]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, HeaderRowDirective, HeaderRowsDirective, TimelineViews,
  Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 11, 31)}
    eventSettings={eventSettings} startHour='09:00' endHour='13:00'>
    <HeaderRowsDirective>
      <HeaderRowDirective option='Year' />
      <HeaderRowDirective option='Month' />
      <HeaderRowDirective option='Week' />
      <HeaderRowDirective option='Date' />
      <HeaderRowDirective option='Hour' />
    </HeaderRowsDirective>
```

```

    <ViewsDirective>
      <ViewDirective option='TimelineWeek' />
    </ViewsDirective>
    <Inject services={[TimelineViews]} />
  </ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>

```



```

        <div id='loader'>Loading....</div>
      </div>
</body>
</html>

```

Importing `HeaderRowsDirective` and `HeaderRowDirective` is mandatory.

Display year and month rows in timeline views

To display the timeline Scheduler simply with year and month names alone, define the option `Year` and `Month` within the `HeaderRowDirective`.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, HeaderRowDirective, HeaderRowsDirective,
TimelineMonth, Inject, ViewsDirective, ViewDirective } from '@syncfusion/ej2-
react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 11, 31)} eventSettings={eventSettings}>
    <HeaderRowsDirective>
      <HeaderRowDirective option='Year' />
      <HeaderRowDirective option='Month' />
    </HeaderRowsDirective>
    <ViewsDirective>
      <ViewDirective option='TimelineMonth' interval={24} />
    </ViewsDirective>
    <Inject services={[TimelineMonth]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, HeaderRowDirective, HeaderRowsDirective, TimelineMonth,
  Inject,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 11, 31)}
    eventSettings={eventSettings}>
    <HeaderRowsDirective>
      <HeaderRowDirective option='Year' />
      <HeaderRowDirective option='Month' />

```

```

    </HeaderRowsDirective>
    <ViewsDirective>
      <ViewDirective option='TimelineMonth' interval={24} />
    </ViewsDirective>
    <Inject services={[TimelineMonth]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>

```

```

        <div id='schedule'>
            <div id='loader'>Loading....</div>
        </div>
    </body>
</html>

```

Display week numbers in timeline views

The week number can be displayed in a separate header row of the timeline Scheduler by setting **Week** option within **HeaderRowDirective**.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, HeaderRowDirective, HeaderRowsDirective,
TimelineMonth, TimelineViews, Inject, ViewsDirective, ViewDirective } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 11, 31)} eventSettings={eventSettings}>
        <HeaderRowsDirective>
            <HeaderRowDirective option='Week' />
            <HeaderRowDirective option='Date' />
            <HeaderRowDirective option='Hour' />
        </HeaderRowsDirective>
        <ViewsDirective>
            <ViewDirective option='TimelineMonth' interval={24} />
            <ViewDirective option='TimelineWeek' interval={3} />
            <ViewDirective option='TimelineDay' interval={4} />
        </ViewsDirective>
        <Inject services={[TimelineMonth, TimelineViews]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, HeaderRowDirective, HeaderRowsDirective, TimelineMonth,
    TimelineViews, Inject,
    ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 11, 31)}
        eventSettings={eventSettings}>
        <HeaderRowsDirective>

```

```

    <HeaderRowDirective option='Week' />
    <HeaderRowDirective option='Date' />
    <HeaderRowDirective option='Hour' />
  </HeaderRowsDirective>
  <ViewsDirective>
    <ViewDirective option='TimelineMonth' interval={24} />
    <ViewDirective option='TimelineWeek' interval={3} />
    <ViewDirective option='TimelineDay' interval={4} />
  </ViewsDirective>
  <Inject services={[TimelineMonth, TimelineViews]} />
</ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
    }
  </style>

```

```

        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Timeline view displaying dates of a complete year

It is possible to display a complete year in a timeline view by setting **interval** value as 12 and defining **TimelineMonth** view option within the **ViewDirective** of Scheduler.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, HeaderRowDirective, HeaderRowsDirective,
TimelineMonth, Inject, ViewsDirective, ViewDirective } from '@syncfusion/ej2-
react-schedule';
import { eventData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: eventData }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 0, 1)} eventSettings={eventSettings}>
        <HeaderRowsDirective>
            <HeaderRowDirective option='Month' />
            <HeaderRowDirective option='Date' />
        </HeaderRowsDirective>
        <ViewsDirective>
            <ViewDirective option='TimelineMonth' interval={12} />
        </ViewsDirective>
        <Inject services={[TimelineMonth]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, HeaderRowDirective, HeaderRowsDirective, TimelineMonth,
    Inject,
    ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { eventData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: eventData }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 0, 1)}

```

```

    eventSettings={eventSettings}>
    <HeaderRowsDirective>
      <HeaderRowDirective option='Month' />
      <HeaderRowDirective option='Date' />
    </HeaderRowsDirective>
    <ViewsDirective>
      <ViewDirective option='TimelineMonth' interval={12} />
    </ViewsDirective>
    <Inject services={[TimelineMonth]} />
  </ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;

```

```

    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Customizing the header rows using template

You can customize the text of the header rows and display any images or formatted text on each individual header rows using the built-in **template** option available within the **HeaderRowDirective**.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, getWeekNumber, HeaderRowDirective,
HeaderRowsDirective, TimelineMonth, Inject, ViewsDirective, ViewDirective }
from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const eventSettings = { dataSource: scheduleData }
  const instance = new Internationalization();
  const getYearDetails = (value) => {
    return 'Year: ' + instance.formatDate(value.date, { skeleton: 'Y' });
  }
  const getMonthDetails = (value) => {
    return 'Month: ' + instance.formatDate(value.date, { skeleton: 'M'
  });
  }
  const getWeekDetails = (value) => {
    return 'Week ' + getWeekNumber(value.date);
  };
  const yearTemplate = (props) => {
    return (<span className="year">{getYearDetails(props)}</span>);
  }
  const monthTemplate = (props) => {
    return (<span className="month">{getMonthDetails(props)}</span>);
  }
  const weekTemplate = (props) => {
    return (<span className="week">{getWeekDetails(props)}</span>);
  }
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 0, 1)} eventSettings={eventSettings}>
    <HeaderRowsDirective>
      <HeaderRowDirective option='Year' template={yearTemplate} />
      <HeaderRowDirective option='Month' template={monthTemplate} />
      <HeaderRowDirective option='Week' template={weekTemplate} />
      <HeaderRowDirective option='Date' />
    </HeaderRowsDirective>
    <ViewsDirective>
      <ViewDirective option='TimelineMonth' />

```

```

        </ViewsDirective>
        <Inject services={[TimelineMonth]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, getWeekNumber, HeaderRowDirective, HeaderRowsDirective,
    TimelineMonth, Inject,
    ViewsDirective, ViewDirective, CellTemplateArgs
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    const instance: Internationalization = new Internationalization();
    const getYearDetails = (value: CellTemplateArgs) => {
        return 'Year: ' + instance.formatDate((value as CellTemplateArgs).date, {
            skeleton: 'Y' });
    }
    const getMonthDetails = (value: CellTemplateArgs) => {
        return 'Month: ' + instance.formatDate((value as CellTemplateArgs).date, {
            skeleton: 'M' });
    }
    const getWeekDetails = (value: CellTemplateArgs) => {
        return 'Week ' + getWeekNumber((value as CellTemplateArgs).date);
    }
    const yearTemplate = (props): JSX.Element => {
        return (<span className="year">{getYearDetails(props)}</span>);
    }
    const monthTemplate = (props): JSX.Element => {
        return (<span className="month">{getMonthDetails(props)}</span>);
    }
    const weekTemplate = (props): JSX.Element => {
        return (<span className="week">{getWeekDetails(props)}</span>);
    }
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 0, 1)}
        eventSettings={eventSettings}>
        <HeaderRowsDirective>
            <HeaderRowDirective option='Year' template={yearTemplate} />
            <HeaderRowDirective option='Month' template={monthTemplate} />
            <HeaderRowDirective option='Week' template={weekTemplate} />
            <HeaderRowDirective option='Date' />
        </HeaderRowsDirective>
        <ViewsDirective>
            <ViewDirective option='TimelineMonth' />
        </ViewsDirective>
        <Inject services={[TimelineMonth]} />
    </ScheduleComponent>)

```



```
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Row auto height in React Schedule component

By default, the height of the Scheduler rows in Timeline views are static and therefore, when the same time range holds multiple overlapping appointments, a **+n more** text indicator will be displayed. With this feature enabled, you can now view all the overlapping appointments present in those specific time range by auto-adjusting the row height based on the presence of the appointments count, instead of displaying the **+n more** text indicators.

To enable auto row height adjustments on Scheduler Timeline views and Month view, set **true** to the **rowAutoHeight** property whose default value is **false**.

This auto row height adjustment is applicable only on all the Timeline views as well as on the calendar Month view.

Now, let's see how it works on those applicable views with examples.

Calendar month view

By default, the rows of the calendar Month view can hold only the limited appointments count based on its row height, and the rest of the overlapping appointments are indicated with a **+n more** text indicator. The following example shows how the month view row auto-adjusts based on the number of appointments count, when this **RowAutoHeight** feature is enabled.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Month, Inject, ViewsDirective, ViewDirective }
from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} rowAutoHeight={true}>
    <ViewsDirective>
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Month]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Month, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
```

```

const eventSettings: EventSettingsModel = { dataSource: scheduleData };
return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} rowAutoHeight={true}>
  <ViewsDirective>
    <ViewDirective option='Month' />
  </ViewsDirective>
  <Inject services={[Month]} />
</ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>

```

```
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

Timeline views

When the feature **RowAutoHeight** is enabled in Timeline views, the row height gets auto-adjusted based on the number of overlapping events occupied on the same time range, which is demonstrated in the following example.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, TimelineViews, Inject, TimelineMonth, Agenda,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} rowAutoHeight={true}>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineWorkWeek' />
      <ViewDirective option='TimelineMonth' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[TimelineViews, TimelineMonth, Agenda]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, TimelineViews, Inject, TimelineMonth, Agenda,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} rowAutoHeight={true}>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineWorkWeek' />
      <ViewDirective option='TimelineMonth' />
    </ViewsDirective>
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

```

    <ViewDirective option='Agenda' />
  </ViewsDirective>
  <Inject services={[TimelineViews, TimelineMonth, Agenda]} />
</ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>

```

```
</body>
</html>
```

Timeline views with multiple resources

The following example shows how the auto row adjustment feature works on timeline views with multiple resources.

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { TimelineViews, ScheduleComponent, ViewsDirective, ViewDirective,
ResourcesDirective, ResourceDirective, Inject, Resize, DragAndDrop } from
'@syncfusion/ej2-react-schedule';
import { roomData } from './datasource';
const App = () => {
  const ownerData = [
    { text: 'Room A', id: 1, color: '#98AFC7' },
    { text: 'Room B', id: 2, color: '#99c68e' },
    { text: 'Room C', id: 3, color: '#C2B280' },
    { text: 'Room D', id: 4, color: '#3090C7' },
    { text: 'Room E', id: 5, color: '#95b9' },
    { text: 'Room F', id: 6, color: '#95b9c7' },
    { text: 'Room G', id: 7, color: '#deb887' },
    { text: 'Room H', id: 8, color: '#3090C7' },
    { text: 'Room I', id: 9, color: '#98AFC7' },
    { text: 'Room J', id: 10, color: '#778899' }
  ];
  const fieldsData = {
    id: 'Id',
    subject: { title: 'Summary', name: 'Subject' },
    location: { title: 'Location', name: 'Location' },
    description: { title: 'Comments', name: 'Description' },
    startTime: { title: 'From', name: 'StartTime' },
    endTime: { title: 'To', name: 'EndTime' }
  }
  const eventSettings = { dataSource: roomData, fields: fieldsData };
  const group = { enableCompactView: false, resources: ['MeetingRoom'] };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 7, 1)} rowAutoHeight={true} eventSettings={eventSettings}
group={group}>
    <ResourcesDirective>
      <ResourceDirective field='RoomId' title='Room Type' name='MeetingRoom'
allowMultiple={true} dataSource={ownerData} textField='text' idField='id'
colorField='color'>
      </ResourceDirective></ResourcesDirective>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='TimelineWeek' />
    </ViewsDirective>
    <Inject services={[TimelineViews, Resize, DragAndDrop]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
  TimelineViews, ScheduleComponent, ViewsDirective,
  ViewDirective, ResourcesDirective, ResourceDirective, Inject, Resize,
  DragAndDrop, EventSettingsModel, GroupModel
} from '@syncfusion/ej2-react-schedule';
import { roomData } from '../datasource';
const App = () => {
  const ownerData: Object[] = [
    { text: 'Room A', id: 1, color: '#98AFC7' },
    { text: 'Room B', id: 2, color: '#99c68e' },
    { text: 'Room C', id: 3, color: '#C2B280' },
    { text: 'Room D', id: 4, color: '#3090C7' },
    { text: 'Room E', id: 5, color: '#95b9' },
    { text: 'Room F', id: 6, color: '#95b9c7' },
    { text: 'Room G', id: 7, color: '#deb887' },
    { text: 'Room H', id: 8, color: '#3090C7' },
    { text: 'Room I', id: 9, color: '#98AFC7' },
    { text: 'Room J', id: 10, color: '#778899' }
  ];
  const fieldsData = {
    id: 'Id',
    subject: { title: 'Summary', name: 'Subject' },
    location: { title: 'Location', name: 'Location' },
    description: { title: 'Comments', name: 'Description' },
    startTime: { title: 'From', name: 'StartTime' },
    endTime: { title: 'To', name: 'EndTime' }
  };
  const eventSettings: EventSettingsModel = { dataSource: roomData, fields:
fieldsData };
  const group: GroupModel = { enableCompactView: false, resources:
['MeetingRoom'] };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 7, 1)} rowAutoHeight={true} eventSettings={eventSettings}
group={group} >
    <ResourcesDirective>
      <ResourceDirective field='RoomId' title='Room Type' name='MeetingRoom'
allowMultiple={true}
        dataSource={ownerData} textField='text' idField='id'
colorField='color'>
      </ResourceDirective></ResourcesDirective>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='TimelineWeek' />
    </ViewsDirective>
    <Inject services={[TimelineViews, Resize, DragAndDrop]} />
    </ScheduleComponent>
  </>);
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Appointments occupying entire cell

By default, with the feature **rowAutoHeight**, there will be a space in the bottom of the cell when appointment is rendered. To avoid this space, we can set true to the property **ignoreWhitespace** with

in `eventSettings` whereas its default property value is false. In the following code example, the whitespace below the appointments has been ignored.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { TimelineViews, TimelineMonth, ScheduleComponent, ViewsDirective,
ViewDirective, ResourcesDirective, ResourceDirective, Inject } from
'@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: resourceData, ignoreWhitespace: true };
  const group = { resources: ['Rooms', 'Owners'] };
  const roomData = [
    { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
    { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
  ];
  const ownerData = [
    { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
  ];
  return (<ScheduleComponent width='100%' height='550px'
currentView='TimelineWeek' rowAutoHeight={true} selectedDate={new Date(2021,
7, 4)} eventSettings={eventSettings} group={group}>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='RoomId' title='Room' name='Rooms'
dataSource={roomData} textField='RoomText' idField='Id'
colorField='RoomColor'>
      </ResourceDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={ownerData} textField='OwnerText'
idField='Id' groupIdField='GroupId' colorField='OwnerColor'>
      </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[TimelineViews, TimelineMonth]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  TimelineViews, TimelineMonth, ScheduleComponent, ViewsDirective,
  ViewDirective, EventSettingsModel, GroupModel,
  ResourcesDirective, ResourceDirective, Inject
}
```

```

} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: resourceData,
ignoreWhitespace: true };
  const group: GroupModel = { resources: ['Rooms', 'Owners'] };
  const roomData: Object[] = [
    { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
    { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
  ]
  const ownerData: Object[] = [
    { OwnerText: 'Nancy', Id: 1, GroupId: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, GroupId: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, GroupId: 1, OwnerColor: '#7499e1' }
  ];
  return (<ScheduleComponent width='100%' height='550px'
currentView='TimelineWeek' rowAutoHeight={true} selectedDate={new Date(2021,
7, 4)} eventSettings={eventSettings} group={group}>
    <ViewsDirective>
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='RoomId' title='Room' name='Rooms'
        dataSource={roomData} textField='RoomText' idField='Id'
colorField='RoomColor'>
      </ResourceDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true}
        dataSource={ownerData} textField='OwnerText' idField='Id'
groupIDField='GroupId' colorField='OwnerColor'>
      </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[TimelineViews, TimelineMonth]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Note: The property `ignoreWhitespace` will be applicable only when `rowAutoHeight` feature is enabled in the Scheduler

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Header bar in React Schedule component

The header part of Scheduler can be customized easily with the built-in options available.

Show or Hide header bar

By default, the header bar holds the date and view navigation options, through which the user can switch between the dates and various views. This header bar can be hidden from the UI by setting `false` to the `showHeaderBar` property. Its default value is `true`.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';

```

```
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} showHeaderBar={false} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Inject, ViewsDirective,
  ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} showHeaderBar={false} eventSettings={eventSettings} >
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Customizing header bar

Apart from the default date navigation and view options available on the header bar, you can add custom items into the Scheduler header bar by making use of the `actionBegin` event. Here, an employee image is added to the header bar, clicking on which will open the popup showing that person's short profile information.

INDEX.JSX

```

import { useRef, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, ViewsDirective, ViewDirective, Month, Inject }
from '@syncfusion/ej2-react-schedule';
import { createElement, compile } from '@syncfusion/ej2-base';
import { Popup } from '@syncfusion/ej2-popups';
import { scheduleData } from '../datasource';
const App = () => {
    const schedule = useRef(null);

```

```

const eventSettings = { dataSource: scheduleData };
let profilePopup;
const onActionBegin = (args) => {
    if (args.requestType === 'toolbarItemRendering') {
        const userIconItem = {
            align: 'Right', prefixIcon: 'user-icon', text: 'Nancy',
            cssClass: 'e-schedule-user-icon'
        };
        args.items.push(userIconItem);
    }
    const onActionComplete = (args) => {
        const scheduleElement = document.getElementById('schedule');
        if (args.requestType === 'toolBarItemRendered' && scheduleElement) {
            const userIconEle = scheduleElement.querySelector('.e-schedule-user-icon');
            if (userIconEle) {
                userIconEle.onclick = () => {
                    if (profilePopup) {
                        profilePopup.relateTo = userIconEle;
                        profilePopup.dataBind();
                        if (profilePopup.element.classList.contains('e-popup-close')) {
                            profilePopup.show();
                        } else {
                            profilePopup.hide();
                        }
                    }
                };
            }
        }
    }
    useEffect(() => {
        const scheduleElement = schedule.current.element;
        let profilePopup;
        if (scheduleElement) {
            const userContentEle = createElement('div', {
                className: 'e-profile-wrapper'
            });
            scheduleElement.parentElement?.appendChild(userContentEle);
            const userIconEle = scheduleElement.querySelector('.e-schedule-user-icon');
            const getDOMString = compile('<div class="profile-container"><div class="profile-image">' +
                '</div><div class="content-wrap"><div class="name">Nancy</div>' +
                '<div class="destination">Product Manager</div><div class="status">' +
                '<div class="status-icon"></div>Online</div></div></div>');
            const output = getDOMString({});
            profilePopup = new Popup(userContentEle, {
                content: output[0],
                relateTo: userIconEle,
                position: { X: 'left', Y: 'bottom' },
                collision: { X: 'flip', Y: 'flip' },
                targetType: 'relative',
                viewPortElement: scheduleElement,

```

```

        width: 185,
        height: 80
    });
    profilePopup.hide();
}
}, []);
return (<ScheduleComponent cssClass='schedule-header-bar' width='100%'
height='550px' ref={schedule}
    selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
    actionBegin={onActionBegin} actionComplete={onActionComplete}>
    <ViewsDirective>
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Month]} />
    </ScheduleComponent>)
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, ViewsDirective, ViewDirective, Month, Inject,
    ActionEventArgs, ToolbarActionArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { createElement, compile } from '@syncfusion/ej2-base';
import { ItemModel } from '@syncfusion/ej2-react-navigations';
import { Popup } from '@syncfusion/ej2-popups';
import { scheduleData } from '../datasource';
const App = () => {
    const schedule = useRef<ScheduleComponent>(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    let profilePopup: Popup;
    const onActionBegin = (args: ActionEventArgs & ToolbarActionArgs): void => {
        if (args.requestType === 'toolbarItemRendering') {
            const userIconItem: ItemModel = {
                align: 'Right', prefixIcon: 'user-icon', text: 'Nancy', cssClass: 'e-
schedule-user-icon'
            };
            args.items.push(userIconItem);
        }
    }
    const onActionComplete = (args: ActionEventArgs): void => {
        const scheduleElement: HTMLElement = document.getElementById('schedule');
        if (args.requestType === 'toolBarItemRendered' && scheduleElement) {
            const userIconEle: HTMLElement = scheduleElement.querySelector('.e-
schedule-user-icon');
            if (userIconEle) {
                userIconEle.onclick = () => {
                    if (profilePopup) {
                        profilePopup.relateTo = userIconEle;
                        profilePopup.dataBind();
                    }
                }
            }
        }
    }
};

```

```

        if (profilePopup.element.classList.contains('e-popup-close')) {
            profilePopup.show();
        } else {
            profilePopup.hide();
        }
    }
};
}
}
}
useEffect(() => {
    const scheduleElement: HTMLElement = schedule.current.element;
    let profilePopup: Popup;
    if (scheduleElement) {
        const userContentEle: HTMLElement = createElement('div', {
            className: 'e-profile-wrapper'
        });
        scheduleElement.parentElement.appendChild(userContentEle);
        const userIconEle: HTMLElement | null =
scheduleElement.querySelector('.e-schedule-user-icon');
        const getDOMString: (data: object) => NodeList = compile('<div
class="profile-container"><div class="profile-image">' +
            '</div><div class="content-wrap"><div class="name">Nancy</div>' +
            '<div class="destination">Product Manager</div><div class="status">'
+
            '<div class="status-icon"></div>Online</div></div></div>');
        const output: NodeList = getDOMString({});
        profilePopup = new Popup(userContentEle, {
            content: output[0] as HTMLElement,
            relateTo: userIconEle,
            position: { X: 'left', Y: 'bottom' },
            collision: { X: 'flip', Y: 'flip' },
            targetType: 'relative',
            viewPortElement: scheduleElement,
            width: 185,
            height: 80
        });
        profilePopup.hide();
    }
}, []);
return (<ScheduleComponent cssClass='schedule-header-bar' width='100%'
height='550px' ref={schedule}
    selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
    actionBegin={onActionBegin} actionComplete={onActionComplete}>
    <ViewsDirective>
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Month]} />
</ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```
<!DOCTYPE html>
```



```

<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

How to display the view options within the header bar popup

By default, the header bar holds the view navigation options, through which the user can switch between various views. You can move this view options to the header bar popup by setting `true` to the `enableAdaptiveUI` property.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='500px' selectedDate={new Date(2018, 1, 15)} enableAdaptiveUI={true} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='500px' selectedDate={new Date(2018, 1, 15)} enableAdaptiveUI={true} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-inputs/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Refer [here](#) to know more about adaptive UI in resources scheduler.

Date header customization

The Scheduler UI that displays the date text on all views are considered as the date header cells. You can customize the date header cells of Scheduler either using `dateHeaderTemplate` or `renderCell` event.

Using date header template

The `dateHeaderTemplate` option is used to customize the date header cells of day, week, work-week and timeline views.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, Inject, TimelineViews } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const instance = new Internationalization();
    const getDateHeaderText = (value) => {
        return instance.formatDate(value, { skeleton: 'Ed' });
    }
}

```

```

const getWeather = (value) => {
  switch (value.getDay()) {
    case 0:
      return '<div class="weather-text">25°C</div>';
    case 1:
      return '<div class="weather-text">18°C</div>';
    case 2:
      return '<div class="weather-text">10°C</div>';
    case 3:
      return '<div class="weather-text">16°C</div>';
    case 4:
      return '<div class="weather-text">8°C</div>';
    case 5:
      return '<div class="weather-text">27°C</div>';
    case 6:
      return '<div class="weather-text">17°C</div>';
    default:
      return null;
  }
}

const dateHeaderTemplate = (props) => {
  return (<div><div>{getDateHeaderText(props.date)}</div><div
className="date-text" dangerouslySetInnerHTML={{ __html:
getWeather(props.date) }}></div></div>);
}

return (<ScheduleComponent width='100%' height='550px'
cssClass='schedule-date-header-template' selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings} dateHeaderTemplate={dateHeaderTemplate}>
  <ViewsDirective>
    <ViewDirective option='Day' />
    <ViewDirective option='Week' />
    <ViewDirective option='WorkWeek' />
    <ViewDirective option='TimelineWeek' />
  </ViewsDirective>
  <Inject services={[Day, Week, WorkWeek, TimelineViews]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, Inject, TimelineViews } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  const instance: Internationalization = new Internationalization();
  const getDateHeaderText = (value: Date): string => {
    return instance.formatDate(value, { skeleton: 'Ed' });
  }
  const getWeather = (value: Date) => {

```

```

switch (value.getDay()) {
  case 0:
    return '<div class="weather-text">25°C</div>';
  case 1:
    return '<div class="weather-text">18°C</div>';
  case 2:
    return '<div class="weather-text">10°C</div>';
  case 3:
    return '<div class="weather-text">16°C</div>';
  case 4:
    return '<div class="weather-text">8°C</div>';
  case 5:
    return '<div class="weather-text">27°C</div>';
  case 6:
    return '<div class="weather-text">17°C</div>';
  default:
    return null;
}
}
const dateHeaderTemplate = (props): JSX.Element => {
  return (<div><div>{getDateHeaderText(props.date)}</div><div
className="date-text" dangerouslySetInnerHTML={{ __html:
getWeather(props.date)}></div></div>);
}
return (<ScheduleComponent width='100%' height='550px' cssClass='schedule-
date-header-template'
  selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
  dateHeaderTemplate={dateHeaderTemplate}>
  <ViewsDirective>
    <ViewDirective option='Day' />
    <ViewDirective option='Week' />
    <ViewDirective option='WorkWeek' />
    <ViewDirective option='TimelineWeek' />
  </ViewsDirective>
  <Inject services={[Day, Week, WorkWeek, TimelineViews]} />
</ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

INDEX.CSS

```

/* csslint ignore:start */
.schedule-date-header-template.e-schedule .e-vertical-view .e-header-cells {
    padding: 0;
    text-align: center !important;
}
.schedule-date-header-template.e-schedule .date-text {
    font-size: 14px;
}
.schedule-date-header-template.e-schedule.e-device .date-text {
    font-size: 12px;
}
.schedule-date-header-template.e-schedule .weather-text {
    font-size: 11px;
}
.schedule-date-header-template.e-schedule .e-month-view .weather-text {
    float: right;
    margin: -20px 2px 0 0;
    width: 20px;
}

```

```

    height: 20px;
  }
  /* csslint ignore:end */

```

Using renderCell event

In month view, the date header template is not applicable and therefore the same customization can be added beside the date text in month cells by making use of the `renderCell` event.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Month, Inject }
from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  const getWeather = (value) => {
    switch (value.getDay()) {
      case 0:
        return '<div class="weather-text">25°C</div>';
      case 1:
        return '<div class="weather-text">18°C</div>';
      case 2:
        return '<div class="weather-text">10°C</div>';
      case 3:
        return '<div class="weather-text">16°C</div>';
      case 4:
        return '<div class="weather-text">8°C</div>';
      case 5:
        return '<div class="weather-text">27°C</div>';
      case 6:
        return '<div class="weather-text">17°C</div>';
      default:
        return null;
    }
  }
  const onRenderCell = (args) => {
    if (args.elementType === 'monthCells') {
      let ele = document.createElement('div');
      ele.innerHTML = getWeather(args.date);
      (args.element).appendChild(ele.firstChild);
    }
  }
  return (<ScheduleComponent width='100%' height='550px'
    cssClass='schedule-date-header-template' renderCell={onRenderCell}
    selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Month]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, View, Month,
RenderCellEventArgs, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  const getWeather = (value: Date) => {
    switch (value.getDay()) {
      case 0:
        return '<div class="weather-text">25°C</div>';
      case 1:
        return '<div class="weather-text">18°C</div>';
      case 2:
        return '<div class="weather-text">10°C</div>';
      case 3:
        return '<div class="weather-text">16°C</div>';
      case 4:
        return '<div class="weather-text">8°C</div>';
      case 5:
        return '<div class="weather-text">27°C</div>';
      case 6:
        return '<div class="weather-text">17°C</div>';
      default:
        return null;
    }
  }
  const onRenderCell = (args: RenderCellEventArgs): void => {
    if (args.elementType === 'monthCells') {
      let ele: Element = document.createElement('div');
      ele.innerHTML = getWeather(args.date);
      (args.element).appendChild(ele.firstChild);
    }
  }
  return (<ScheduleComponent width='100%' height='550px' cssClass='schedule-
date-header-template'
    renderCell={onRenderCell} selectedDate={new Date(2018, 1, 15)}
    eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Month]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>

```



```

<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta name="description" content="Essential JS 2 for React Components" />
<meta name="author" content="Syncfusion" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

INDEX.CSS

```

/* csslint ignore:start */
.schedule-date-header-template.e-schedule .e-vertical-view .e-header-cells {
    padding: 0;
    text-align: center !important;
}
.schedule-date-header-template.e-schedule .date-text {
    font-size: 14px;
}

```

```
.schedule-date-header-template.e-schedule.e-device .date-text {
  font-size: 12px;
}
.schedule-date-header-template.e-schedule .weather-text {
  font-size: 11px;
}
.schedule-date-header-template.e-schedule .e-month-view .weather-text {
  float: right;
  margin: -20px 2px 0 0;
  width: 20px;
  height: 20px;
}
/* csslint ignore:end */
```

Customizing the date range text

The [dateRangeTemplate](#) option allows you to customize the text content of the date range displayed in the scheduler. By default, the date range text is determined by the scheduler view being used. However, you can use the [dateRangeTemplate](#) option to override the default text and specify your own custom text to be displayed.

The [dateRangeTemplate](#) property includes `startDate`, `endDate` and `currentView` options, you can customize the date range text using these available options.

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, Inject, TimelineViews } from '@syncfusion/ej2-react-schedule';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const instance = new Internationalization();
  const getDateRange = (startDate, endDate) => {
    return instance.formatDate(startDate, { skeleton: 'yMd' }) + ' - ' +
instance.formatDate(endDate, { skeleton: 'yMd' });
  }
  const dateRangeTemplate = (props) => {
    return (<div>{getDateRange(props.startDate, props.endDate)}</div>);
  }
  return (<ScheduleComponent width='100%' height='550px'
dateRangeTemplate={dateRangeTemplate}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='TimelineWeek' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, TimelineViews]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { ScheduleComponent, ViewsDirective, ViewDirective, View, Day, Week,
WorkWeek, Month, RenderCellEventArgs, Inject, TimelineViews } from
'@syncfusion/ej2-react-schedule';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const instance: Internationalization = new Internationalization();
  const getDateRange = (startDate: Date, endDate: Date): string => {
    return instance.formatDate(startDate, { skeleton: 'yMd' }) + ' - ' +
instance.formatDate(endDate, { skeleton: 'yMd' });
  }
  const dateRangeTemplate = (props): JSX.Element => {
    return (<div>{getDateRange(props.startDate, props.endDate)}</div>);
  }
  return (
    <ScheduleComponent width='100%' height='550px'
      dateRangeTemplate={dateRangeTemplate}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='TimelineWeek' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, TimelineViews]} />
    </ScheduleComponent>
  )
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

INDEX.CSS

```

/* csslint ignore:start */
.schedule-date-header-template.e-schedule .e-vertical-view .e-header-cells {
    padding: 0;
    text-align: center !important;
}
.schedule-date-header-template.e-schedule .date-text {
    font-size: 14px;
}
.schedule-date-header-template.e-schedule.e-device .date-text {
    font-size: 12px;
}
.schedule-date-header-template.e-schedule .weather-text {
    font-size: 11px;
}
.schedule-date-header-template.e-schedule .e-month-view .weather-text {
    float: right;
    margin: -20px 2px 0 0;
    width: 20px;
    height: 20px;
}
/* csslint ignore:end */

```

Customizing header indent cells

It is possible to customize the header indent cells using the `headerIndentTemplate` option and change the look and appearance in both the vertical and timeline views. In vertical views, You can customize the

header indent cells at the hierarchy level and you can customize the resource header left indent cell in timeline views using the template option.

Example: To customize the header left indent cell to display resources text, refer to the below code example.

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { Week, TimelineViews, TimelineMonth, Day, ScheduleComponent,
ViewsDirective, ViewDirective, ResourcesDirective, ResourceDirective, Inject
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: resourceData };
  const group = { resources: ['Owners'] };
  const ownerData = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ];
  const headerIndentTemplate = () => {
    return (<div className='e-resource-text'>
      <div className="text">Resources</div></div>);
  }
  return (<ScheduleComponent width='100%' height='550px' currentView='Week'
headerIndentTemplate={headerIndentTemplate} selectedDate={new Date(2018, 3,
1)} eventSettings={eventSettings} group={group}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={ownerData} textField='OwnerText'
idField='Id' colorField='OwnerColor'>
        </ResourceDirective>
      </ResourcesDirective>
      <Inject services={[Day, Week, TimelineViews, TimelineMonth]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import {
  Week, TimelineViews, TimelineMonth, Day, ScheduleComponent, GroupModel,
ViewsDirective, ViewDirective, ResourcesDirective, EventSettingsModel,
ResourceDirective, Inject
} from '@syncfusion/ej2-react-schedule';
```

```

import { resourceData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: resourceData };
  const group: GroupModel = { resources: ['Owners'] };
  const ownerData: object[] = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ];
  const headerIndentTemplate = () => {
    return (
      <div className='e-resource-text'>
        <div className="text">Resources</div></div>
    );
  }
  return (<ScheduleComponent width='100%' height='550px' currentView='Week'
headerIndentTemplate={headerIndentTemplate} selectedDate={new Date(2018, 3,
1)} eventSettings={eventSettings} group={group}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='TimelineMonth' />
    </ViewsDirective>
    <ResourcesDirective>
      <ResourceDirective field='OwnerId' title='Owner' name='Owners'
allowMultiple={true} dataSource={ownerData} textField='OwnerText'
idField='Id' colorField='OwnerColor'>
    </ResourceDirective>
    </ResourcesDirective>
    <Inject services={[Day, Week, TimelineViews, TimelineMonth]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Timescale in React Schedule component

The time slots are usually the time cells that are displayed on the Day, Week and Work Week views of both the calendar (to the left most position) and timeline views (at the top position). The **timeScale** property allows you to control and set the required time slot duration for the work cells displayed on Scheduler. It includes the following sub-options such as,

- **enable** - When set to **true**, allows the Scheduler to display the appointments accurately against the exact time duration. If set to **false**, all the appointments of a day will be displayed one below the other with no gridlines displayed. Its default value is **true**.
- **interval** – Defines the time duration on which the time axis to be displayed either in 1 hour or 30 minutes interval and so on. It accepts the values in minutes and defaults to 60.
- **slotCount** – Decides the number of slot count to be split for the specified time interval duration. It defaults to 2, thus displaying two slots to represent an hour(each slot depicting 30 minutes duration).

Note: The upper limit for rendering slots within a single day, utilizing the **interval** and **slotCount** properties of the **timeScale**, stands at 1000. This constraint aligns with the maximum **colspan** value

permissible for the **table** element, also capped at 1000. This particular restriction is relevant exclusively to the **TimelineDay**, **TimelineWeek** and **TimelineWorkWeek** views.

Setting different time slot duration

The **interval** and **slotCount** properties can be used together on the Scheduler to set different time slot duration which is depicted in the following code example. Here, six time slots together represents an hour.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Day, Week, WorkWeek, TimelineViews, ScheduleComponent,
ViewsDirective, ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  const timeScale = { enable: true, interval: 60, slotCount: 6 };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} timeScale={timeScale}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='TimelineDay' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, TimelineViews]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  Day, Week, WorkWeek, TimelineViews, ScheduleComponent, ViewsDirective,
  ViewDirective, Inject, EventSettingsModel, TimeScalesModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const timeScale: TimeScalesModel = { enable: true, interval: 60, slotCount:
6 };
  return <ScheduleComponent width='100%' height='550px'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
timeScale={timeScale}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='TimelineDay' />
    </ViewsDirective>
  </ScheduleComponent>;
}
```



```

    <Inject services={[Day, Week, WorkWeek, TimelineViews]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Customizing time cells using template

The `timeScale` property also provides template option to allow customization of time slots which are as follows,

- **majorSlotTemplate** - The template option to be applied for major time slots. Here, the template accepts either the string or `HTMLElement` as template design and then the parsed design is displayed onto the time cells. The time details can be accessed within this template.
- **minorSlotTemplate** - The template option to be applied for minor time slots. Here, the template accepts either the string or `HTMLElement` as template design and then the parsed design is displayed onto the time cells. The time details can be accessed within this template.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, TimelineViews, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const eventSettings = { dataSource: data };
  const instance = new Internationalization();
  const majorSlotTemplate = (props) => {
    return (<div>{instance.formatDate(props.date, { skeleton: 'hm'
  })}</div>);
  }
  const minorSlotTemplate = (props) => {
    return (<div style={{ textAlign: 'right', marginRight: '15px' }}>
      {instance.formatDate(props.date, { skeleton: 'ms' }).replace(':00',
      '')}</div>);
  }
  const timeScale = {
    enable: true, interval: 60, slotCount: 6, majorSlotTemplate:
majorSlotTemplate.bind(this),
    minorSlotTemplate: minorSlotTemplate.bind(this)
  };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} timeScale={timeScale}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='WorkWeek' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, TimelineViews]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, ViewsDirective, ViewDirective,
  Day, Week, WorkWeek, TimelineViews, Inject, EventSettingsModel,
  TimeScaleModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const instance = new Internationalization();
  const minorSlotTemplate = (props): JSX.Element => {
    return (<div style={{ textAlign: 'right', marginRight: '15px' }}>
      {instance.formatDate(props.date, { skeleton: 'ms' }).replace(':00',
        '')}</div>);
  }
  const majorSlotTemplate = (props): JSX.Element => {
    return (<div>{instance.formatDate(props.date, { skeleton: 'hm'
  })}</div>);
  }
  const timeScale: TimeScaleModel = {
    enable: true, interval: 60, slotCount: 6, majorSlotTemplate:
majorSlotTemplate.bind(this),
    minorSlotTemplate: minorSlotTemplate.bind(this)
  };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} timeScale={timeScale}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='WorkWeek' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, TimelineViews]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Hide the timescale

The grid lines which indicates the exact time duration can be enabled or disabled on the Scheduler, by setting `true` or `false` to the `enable` option within the `timeScale` property. It's default value is `true`.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Day, Week, WorkWeek, TimelineViews, ScheduleComponent,
ViewsDirective, ViewDirective, Inject } from '@syncfusion/ej2-react-
schedule';
import { scheduleData } from '../datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const timeScale = { enable: false };
    return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} timeScale={timeScale}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='TimelineWeek' />

```

```

    <ViewDirective option='WorkWeek' />
  </ViewsDirective>
  <Inject services={[Day, Week, WorkWeek, TimelineViews]} />
</ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  Day, Week, WorkWeek, TimelineViews, ScheduleComponent, ViewsDirective,
  ViewDirective, Inject, EventSettingsModel, TimeScaleModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const timeScale: TimeScaleModel = { enable: false };
  return <ScheduleComponent width='100%' height='550px'
    selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
    timeScale={timeScale}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='WorkWeek' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, TimelineViews]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Highlighting current date and time

By default, Scheduler indicates current date with a highlighted date header on all views, as well as marks accurately the system's current time on specific views such as Day, Week, Work Week, Timeline Day, Timeline Week and Timeline Work Week views. To stop highlighting the current time indicator on Scheduler views, set `false` to the `showTimeIndicator` property which defaults to `true`.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Day, Week, TimelineViews, ScheduleComponent, ViewsDirective,
ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
const App = () => {
    return (<ScheduleComponent width='100%' height='550px'
showTimeIndicator={true}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='TimelineWeek' />
        </ViewsDirective>
        <Inject services={[Day, Week, TimelineViews]} />
    </ScheduleComponent>);
}
;

```

```
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  Day, Week, TimelineViews, ScheduleComponent, ViewsDirective, ViewDirective,
  Inject
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  return (<ScheduleComponent width='100%' height='550px'
    showTimeIndicator={true}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='TimelineWeek' />
      </ViewsDirective>
      <Inject services={[Day, Week, TimelineViews]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
```

```

<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Working days in React Schedule component

The Scheduler can be customized on various aspects as well as it inherits almost all the calendar-specific features such as options,

- To set custom time range display on Scheduler
- To set different working hours
- To set different working days
- To set different first day of week
- To show/hide weekend days
- To show the week number
- To display the current time indicator

Set working days

By default, Scheduler considers the week days from Monday to Friday as **Working days** and therefore defaults to [1,2,3,4,5] - where 1 represents Monday, 2 represents Tuesday and so on. The days which are not defined in this working days collection are considered as non-working days. Therefore, when the weekend days are set to hide from Scheduler, all those non-working days too get hidden from the layout.

The Work week and Timeline Work week views display exactly the defined working days on Scheduler layout, whereas other views display all the days and simply differentiate the non-working days on UI with inactive cell color.

The working or business hours depiction on Scheduler are usually valid only on these specified working days.

The following example code depicts how to set the Scheduler to display Monday, Wednesday and Friday as working days of a week.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Week, WorkWeek, TimelineViews, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  const workingDays = [1, 3, 5];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} workDays={workingDays}>
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='TimelineWorkWeek' />
    </ViewsDirective>
    <Inject services={[Week, WorkWeek, TimelineViews]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
ScheduleComponent, Week, WorkWeek, TimelineViews, Inject,
ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const workingDays: number[] = [1, 3, 5];
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} workDays={workingDays} >
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='TimelineWorkWeek' />
    </ViewsDirective>
    <Inject services={[Week, WorkWeek, TimelineViews]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Hiding weekend days

The `showWeekend` property is used to either show or hide the weekend days of a week and it is not applicable on Work week view (as non-working days are usually not displayed on work week view). By default, it is set to `true`. The days which are not a part of the working days collection of a Scheduler are usually considered as non-working or weekend days.

Here, the working days are defined as [1, 3, 4, 5] on Scheduler and therefore the remaining days (0, 2, 6 – Sunday, Tuesday and Saturday) are considered as non-working or weekend days and will be hidden from all the views when `showWeekend` property is set to `false`.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, Month, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} showWeekend={false}
workDays={[1, 3, 4, 5]}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, Month]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, Month, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} showWeekend={false}
workDays={[1, 3, 4, 5]} >
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, Month]} />
  </ScheduleComponent>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Show week numbers

It is possible to show the week number count of a week in the header bar of the Scheduler by setting true to `showWeekNumber` property. By default, its default value is `false`. In Month view, the week numbers are displayed as a first column.

The `showWeekNumber` property is not applicable on Timeline views, as it has the equivalent [headerRows](#) property to handle such requirement with additional customizations.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, Month, Inject, } from
 '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
    Date(2018, 1, 15)} eventSettings={eventSettings} showWeekNumber={true}
    workDays={[1, 3, 4, 5]}>
    <Inject services={[Day, Week, Month]} />
    </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
    Date(2018, 1, 15)} eventSettings={eventSettings} showWeekNumber={true}
    workDays={[1, 3, 4, 5]} >
    <Inject services={[Day, Week, Month]} />
    </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Different options in showing week numbers

By default, week numbers are shown in the Scheduler based on the first day of the year. However, the week numbers can be determined based on the following criteria.

FirstDay – The first week of the year is calculated based on the first day of the year.

FirstFourDayWeek – The first week of the year begins from the first week with four or more days.

FirstFullWeek – The first week of the year begins when meeting the first day of the week (firstDayOfWeek) and the first day of the year.

For more details refer to [this link](#)

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, Month, Inject, } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';

```

```

const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2020, 1, 15)} eventSettings={eventSettings} showWeekNumber={true}
workDays={[1, 3, 4, 5]} weekRule='FirstFourDayWeek'>
    <Inject services={[Day, Week, Month]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2020, 1, 15)} eventSettings={eventSettings} showWeekNumber={true}
    workDays={[1, 3, 4, 5]} weekRule='FirstFourDayWeek'>
    <Inject services={[Day, Week, Month]} />
  </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Note: Enable the `showWeekNumber` property to configure the `weekRule` property. Also, the `weekRule` property depends on the value of the `firstDayOfWeek` property

Set working hours

Working hours indicates the work hour limit within the Scheduler, which is visually highlighted with an active color on work cells. The working hours can be set on Scheduler using the `workHours` property which is of object type and includes the following sub-options,

- `highlight` – enables/disables the highlighting of work hours.
- `start` - sets the start time of the working/business hour of a day.
- `end` - sets the end time limit of the working/business hour of a day.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  const workHours = {
    highlight: true, start: '11:00', end: '20:00'
  };
  return (
    <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} workHours={workHours} eventSettings={eventSettings}>
      <ViewsDirective>

```



```

    <ViewDirective option='Day' />
    <ViewDirective option='Week' />
    <ViewDirective option='WorkWeek' />
  </ViewsDirective>
  <Inject services={[Day, Week, WorkWeek]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel, WorkHoursModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const workHours: WorkHoursModel = {
    highlight: true, start: '11:00', end: '20:00'
  };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} workHours={workHours} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Scheduler displaying custom hours

It is possible to display the event Scheduler layout with specific time durations by hiding the unwanted hours. To do so, set the start and end hour for the Scheduler using the `startHour` and `endHour` properties respectively.

The following code example displays the Scheduler starting from the time range 7.00 AM to 6.00 PM and the remaining hours are hidden on the UI.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} startHour='07:00' endHour='18:00'
eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />

```

```

    <ViewDirective option='WorkWeek' />
  </ViewsDirective>
  <Inject services={[Day, Week, WorkWeek]} />
</ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} startHour='07:00'
endHour='18:00' eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Setting start day of the week

By default, Scheduler defaults to **Sunday** as its first day of a week. To change the Scheduler's start day of a week with different day, set the **firstDayOfWeek** property with the values ranging from 0 to 6.

Here, Sunday is always denoted as 0, Monday as 1 and so on.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Week, Month, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} firstDayOfWeek={1}>
        <ViewsDirective>
            <ViewDirective option='Week' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Week, Month]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Week, Month, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (<ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} firstDayOfWeek={1} >
    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Week, Month]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdn.jsdelivr.net/npm/systemjs/0.19.38/system.js"></scr
ipt>
```

```

<script src="systemjs.config.js"></script>
<style>
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Scroll to specific time and date

You can manually scroll to a specific time on Scheduler by making use of the `scrollTo` method as depicted in the following code example.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { TimePickerComponent } from '@syncfusion/ej2-react-calendars';
import { ScheduleComponent, Day, Week, TimelineViews, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onChange = (args) => {
    scheduleObj.current.scrollTo(args.text);
  }
  return (
    <div className='content-wrapper'>
      <div className='property-section'>
        <table id='property' title='Properties' className='property-panel-
table' style={{ width: '100%' }}>
          <tbody>
            <tr style={{ height: '50px' }}>
              <td style={{ width: '30%' }}>
                <div className='col-md-4' style={{ padding: '8px' }}>
                  Scroll To</div>
              </td>
              <td style={{ width: '70%' }}>
                <div>
                  <TimePickerComponent width={100} value={new Date(2000, 0,
1, 9)} format='HH:mm'
                    change={onChange}></TimePickerComponent>
                </div>
              </td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
  );
}

```

```

        </tr>
      </tbody>
    </table>
  </div>
  <div className='control-section'>
    <ScheduleComponent width='100%' height='550px' ref={scheduleObj}
      selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='TimelineDay' />
        <ViewDirective option='TimelineWeek' />
      </ViewsDirective>
      <Inject services={[Day, Week, TimelineViews]} />
    </ScheduleComponent>
  </div>
</div>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { TimePickerComponent, ChangeEventArgs } from '@syncfusion/ej2-react-calendars';
import {
  ScheduleComponent, Day, Week, TimelineViews, Inject, EventSettingsModel,
  ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const onChange = (args: ChangeEventArgs): void => {
    scheduleObj.current.scrollTo(args.text);
  }
  return (
    <div className='content-wrapper'>
      <div className='property-section'>
        <table id='property' title='Properties' className='property-panel-table' style={{ width: '100%' }}>
          <tbody>
            <tr style={{ height: '50px' }}>
              <td style={{ width: '30%' }}>
                <div className='col-md-4' style={{ padding: '8px' }}>
                  <div>
                    <div>Scroll To</div>
                  </td>
                  <td style={{ width: '70%' }}>
                    <div>
                      <TimePickerComponent width={100} value={new Date(2000, 0, 1, 9)} format='HH:mm'
                        change={onChange}></TimePickerComponent>

```

```

        </div>
      </td>
    </tr>
  </tbody>
</table>
</div>
<div className='control-section'>
  <ScheduleComponent width='100%' height='550px' ref={scheduleObj}
    selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineDay' />
      <ViewDirective option='TimelineWeek' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews]} />
  </ScheduleComponent>
</div>
</div>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>

```



```

<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
</head>
<body>

  <div>
    <div id="schedule">
      <div id='loader'>Loading....</div>
    </div>
  </div>

</body>
</html>

```

How to scroll to current time on initial load

There are scenarios where you may need to load the Scheduler displaying the system's current time on the currently visible view port area. In such cases, the Scheduler needs to be scrolled to a specific time based on the system's current time which is depicted in the following code example.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, TimelineViews, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const instance = new Internationalization();
  const onCreated = () => {
    scheduleObj.current.scrollTo(instance.formatDate(new Date(), { skeleton:
'hm' }));
  }
  return (<ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
created={onCreated}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineWeek' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews]} />
  </ScheduleComponent>);

```

```

}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, TimelineViews, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const instance: Internationalization = new Internationalization();
  const onCreated = (): void => {
    scheduleObj.current.scrollTo(instance.formatDate(new Date(), { skeleton:
'hm' }));
  }
  return (<ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
created={onCreated} >
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineWeek' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008c9f;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

See Also

- [To display the current time indicator](#)
- [To set different working hours dynamically](#)
- [To set different working hours for each resources](#)
- [To set different working days for each resources](#)

Cell customization in React Schedule component

The cells of the Scheduler can be easily customized either using the cell template or `RenderCell` event.

Setting cell dimensions in all views

The height and width of the Scheduler cells can be customized either to increase or reduce its size through the `cssClass` property, which overrides the default CSS applied on cells.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} cssClass='schedule-cell-dimension'
eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
  ActionEventArgs, ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2018, 1, 15)} cssClass='schedule-cell-dimension'
eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
```

```

    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <link href="index.css" rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

INDEX.CSS

```

.templatewrap {
    text-align: center;
    width: 100%;
}

.templatewrap img {
    width: 20px;
    height: 20px;
}

/* csslint ignore:start */

```

```

.schedule-cell-dimension.e-schedule .e-vertical-view .e-date-header-wrap
table col,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-content-wrap table
col {
    width: 200px;
}

.schedule-cell-dimension.e-schedule .e-vertical-view .e-time-cells-wrap
table td,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-work-cells {
    height: 100px;
}

.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells,
.schedule-cell-dimension.e-schedule .e-month-view .e-date-header-wrap table
col {
    width: 200px;
}

.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells {
    height: 200px;
}
/* csslint ignore:end */

```

Check for cell availability

You can check whether the given time range slots are available for event creation or already occupied by other events using the `isSlotAvailable` method. In the following code example, if a specific time slot already contains an appointment, then no more appointments can be added to that cell.

Note: The `isSlotAvailable` is centered around verifying appointments within the present view's date range. Yet, it does not encompass an evaluation of availability for recurrence occurrences that fall beyond this particular date range.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Inject } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const scheduleObj = useRef(null);
    const eventSettings = { dataSource: scheduleData };
    const onActionBegin = (args) => {
        if (args.requestType === 'eventCreate' && args.data.length > 0) {
            let eventData = args.data[0];
            let eventField = scheduleObj.current.eventFields;
            let startDate = eventData[eventField.startTime];
            let endDate = eventData[eventField.endTime];
            args.cancel = !scheduleObj.current.isSlotAvailable(startDate,
            endDate);
        }
    }
}

```

```

    return <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
    selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
    actionBegin={onActionBegin}>
        <Inject services={[Day, Week, WorkWeek]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Inject,
    ActionEventArgs, EventFieldsMapping, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onActionBegin = (args: ActionEventArgs): void => {
        if (args.requestType === 'eventCreate' && (args.data as any).length > 0)
        {
            let eventData: { [key: string]: Object } = args.data[0] as { [key:
            string]: Object };
            let eventField: EventFieldsMapping = scheduleObj.current.eventFields;
            let startDate: Date = eventData[eventField.startTime] as Date;
            let endDate: Date = eventData[eventField.endTime] as Date;
            args.cancel = !scheduleObj.current.isSlotAvailable(startDate, endDate);
        }
    }
    return <ScheduleComponent ref={scheduleObj} width='100%' height='550px'
    selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
    actionBegin={onActionBegin}>
        <Inject services={[Day, Week, WorkWeek]} />
    </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
    base/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Customizing cells in all the views

It is possible to customize the appearance of the cells using both template options and `renderCell` event on all the views.

Using template

The `cellTemplate` option accepts the template string and is used to customize the cell background with specific images or appropriate text on the given date values.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, TimelineViews, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
const App = () => {
    const getMonthCellContent = (date) => {
        if (date.getMonth() === 10 && date.getDate() === 23) {

```



```

        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    } else if (date.getMonth() === 11 && date.getDate() === 9) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/get-together.svg" />';
    } else if (date.getMonth() === 11 && date.getDate() === 13) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    } else if (date.getMonth() === 11 && date.getDate() === 22) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/thanksgiving-day.svg"
/>';
    } else if (date.getMonth() === 11 && date.getDate() === 24) {
        return '';
    } else if (date.getMonth() === 11 && date.getDate() === 25) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/christmas.svg" />';
    } else if (date.getMonth() === 0 && date.getDate() === 1) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/newyear.svg" />';
    } else if (date.getMonth() === 0 && date.getDate() === 14) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    }
    return '';
}
const getWorkCellText = (date) => {
    let weekEnds = [0, 6];
    if (weekEnds.indexOf(date.getDay()) >= 0) {
        return "<img
src='https://ej2.syncfusion.com/demos/src/schedule/images/newyear.svg' />";
    }
    return '';
};
const cellTemplate = (props) => {
    if (props.type === "workCells") {
        return (<div className="templatewrap" dangerouslySetInnerHTML={{
__html: getWorkCellText(props.date) }}></div>);
    }
    if (props.type === "monthCells") {
        return (<div className="templatewrap" dangerouslySetInnerHTML={{
__html: getMonthCellContent(props.date) }}></div>);
    }
    return (<div></div>);
}
return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2017, 11, 15)} cellTemplate={cellTemplate}>
    <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='TimelineWeek' />
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews, Month]} />
</ScheduleComponent>

```

```
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, TimelineViews, Month, Inject,
  ViewsDirective, ViewDirective, CellTemplateArgs
} from '@syncfusion/ej2-react-schedule';
const App = () => {
  const getMonthCellContent = (date: Date) => {
    if (date.getMonth() === 10 && date.getDate() === 23) {
      return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    } else if (date.getMonth() === 11 && date.getDate() === 9) {
      return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/get-together.svg" />';
    } else if (date.getMonth() === 11 && date.getDate() === 13) {
      return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    } else if (date.getMonth() === 11 && date.getDate() === 22) {
      return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/thanksgiving-day.svg"
/>';
    } else if (date.getMonth() === 11 && date.getDate() === 24) {
      return '';
    } else if (date.getMonth() === 11 && date.getDate() === 25) {
      return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/christmas.svg" />';
    } else if (date.getMonth() === 0 && date.getDate() === 1) {
      return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/newyear.svg" />';
    } else if (date.getMonth() === 0 && date.getDate() === 14) {
      return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    }
    return '';
  }
  const getWorkCellText = (date: Date) => {
    let weekEnds: number[] = [0, 6];
    if (weekEnds.indexOf(date.getDay()) >= 0) {
      return '';
    }
    return '';
  };
  const cellTemplate = (props: CellTemplateArgs): JSX.Element => {
    if (props.type === "workCells") {
      return (<div className="templatewrap" dangerouslySetInnerHTML={{
__html: getWorkCellText(props.date) }}></div>);
    }
  }
}
```

```

    if (props.type === "monthCells") {
      return (<div className="templatewrap" dangerouslySetInnerHTML={{
        __html: getMonthCellContent(props.date) }}></div>);
    }
    return (<div></div>);
  }
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
Date(2017, 11, 15)} cellTemplate={cellTemplate}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='TimelineWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, TimelineViews, Month]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {

```

```

        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

INDEX.CSS

```

.templatewrap {
    text-align: center;
    /* In MONTH view the cell template is a SIBLING of event templates. So
    which is necessary to set the parent position relative and the child position
    absolute with 100% width */
    position: absolute;
    width: 100%;
}
.templatewrap img {
    width: 20px;
    height: 20px;
}
/* csslint ignore:start */
.schedule-cell-template.e-schedule .e-month-view .e-work-cells {
    position: relative;
}
.schedule-cell-dimension.e-schedule .e-vertical-view .e-date-header-wrap
table col,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-content-wrap table
col {
    width: 200px;
}
.schedule-cell-dimension.e-schedule .e-vertical-view .e-time-cells-wrap table
td,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-work-cells {
    height: 100px;
}
.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells,
.schedule-cell-dimension.e-schedule .e-month-view .e-date-header-wrap table
col {
    width: 200px;
}
.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells {
    height: 200px;
}
/* csslint ignore:end */

```

Using renderCell event

An alternative to `cellTemplate` is the `renderCell` event, which can also be used to customize the cells with appropriate images or formatted text values.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, Month, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { createElement } from '@syncfusion/ej2-base';
import { scheduleData } from '../datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onRenderCell = (args) => {
        if (args.elementType == 'workCells' || args.elementType ==
'monthCells') {
            let weekEnds = [0, 6];
            if (weekEnds.indexOf((args.date).getDay()) >= 0) {
                let ele = createElement('div', {
                    innerHTML: "<img
src='https://ej2.syncfusion.com/demos/src/schedule/images/newyear.svg' />",
                    className: 'templatewrap'
                });
                (args.element).appendChild(ele);
            }
        }
    }
    return <ScheduleComponent height='550px' currentView='Month'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
renderCell={onRenderCell} cssClass='schedule-cell-template'>
        <ViewsDirective>
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, Month]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, Month, Inject,
    RenderCellEventArgs, ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { createElement } from '@syncfusion/ej2-base';
import { scheduleData } from '../datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onRenderCell = (args: RenderCellEventArgs): void => {
```

```

    if (args.elementType == 'workCells' || args.elementType == 'monthCells')
    {
        let weekEnds: number[] = [0, 6];
        if (weekEnds.indexOf((args.date).getDay()) >= 0) {
            let ele: HTMLElement = createElement('div', {
                innerHTML: "<img
src='https://ej2.syncfusion.com/demos/src/schedule/images/newyear.svg' />",
                className: 'templatewrap'
            });
            (args.element).appendChild(ele);
        }
    }
}

return <ScheduleComponent height='550px' currentView='Month'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
renderCell={onRenderCell}
cssClass='schedule-cell-template'>
    <ViewsDirective>
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, Month]} />
</ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />

```

```

<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

INDEX.CSS

```

.templatewrap {
    text-align: center;
    width: 100%;
}

.templatewrap img {
    width: 20px;
    height: 20px;
}

/* csslint ignore:start */
.schedule-cell-dimension.e-schedule .e-vertical-view .e-date-header-wrap
table col,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-content-wrap table
col {
    width: 200px;
}

.schedule-cell-dimension.e-schedule .e-vertical-view .e-time-cells-wrap
table td,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-work-cells {
    height: 100px;
}

.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells,
.schedule-cell-dimension.e-schedule .e-month-view .e-date-header-wrap table
col {
    width: 200px;
}

```

```

}

.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells {
  height: 200px;
}
/* csslint ignore:end */

```

You can customize cells such as work cells, month cells, all-day cells, header cells, resource header cells using `renderCell` event by checking the `elementType` option within the event. You can check `elementType` with any of the following.

Element type	Description
dateHeader	triggers on header cell rendering.
monthDay	triggers on header cell in month view rendering.
resourceHeader	triggers on resource header cell rendering.
alldayCells	triggers on all day cell rendering.
emptyCells	triggers on empty cell rendering on header bar.
resourceGroupCells	triggers on rendering of work cells for parent resource.
workCells	triggers on work cell rendering.
monthCells	triggers on month cell rendering.
majorSlot	triggers on major time slot cell rendering.
minorSlot	triggers on minor time slot cell rendering.
weekNumberCell	triggers on cell displaying week number.

Customizing cell header in month view

The month header of each date cell in the month view can be customized using the `cellHeaderTemplate` option which accepts the string or `HTMLElement`. The corresponding date can be accessed with the template.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Month, Inject, ViewsDirective, ViewDirective }
from "@syncfusion/ej2-react-schedule";
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const instance = new Internationalization();
  const getDateHeaderText = (props) => {
    return (<div>{instance.formatDate(props.date, { skeleton: "Ed"
  })}</div>);
  }
  return (<ScheduleComponent height='550px'
cellHeaderTemplate={getDateHeaderText}>
  <ViewsDirective>

```



```

    <ViewDirective option='Month' />
  </ViewsDirective>
  <Inject services={[Month]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Month, Inject, ViewsDirective,
  ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { Internationalization } from '@syncfusion/ej2-base';
const App = () => {
  const instance = new Internationalization();
  const getDateHeaderText = (props): JSX.Element => {
    return (<div>{instance.formatDate(props.date, { skeleton: "Ed"
  })}</div>);
  }
  return (<ScheduleComponent height='550px'
cellHeaderTemplate={getDateHeaderText}>
  <ViewsDirective>
    <ViewDirective option='Month' />
  </ViewsDirective>
  <Inject services={[Month]} />
</ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

INDEX.CSS

```

.templatewrap {
    text-align: center;
    width: 100%;
}

.templatewrap img {
    width: 20px;
    height: 20px;
}

/* csslint ignore:start */
.schedule-cell-dimension.e-schedule .e-vertical-view .e-date-header-wrap
table col,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-content-wrap table
col {
    width: 200px;
}

.schedule-cell-dimension.e-schedule .e-vertical-view .e-time-cells-wrap
table td,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-work-cells {
    height: 100px;
}

```

```

}

.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells,
.schedule-cell-dimension.e-schedule .e-month-view .e-date-header-wrap table
col {
    width: 200px;
}

.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells {
    height: 200px;
}
/* csslint ignore:end */

```

Customizing the minimum and maximum date values

Providing the `minDate` and `maxDate` property with some date values, allows the Scheduler to set the minimum and maximum date range. The Scheduler date that lies beyond this minimum and maximum date range will be in a disabled state so that the date navigation will be blocked beyond the specified date range.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Agenda, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    return <ScheduleComponent width='100%' height='550px' currentView='Month'
selectedDate={new Date(2018, 1, 17)} minDate={new Date(2017, 4, 17)}
maxDate={new Date(2018, 5, 17)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
            <ViewDirective option='Agenda' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Agenda, Month]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Agenda, Month, Inject,
    ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';

```

```

const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return <ScheduleComponent width='100%' height='550px' currentView='Month'
    selectedDate={new Date(2018, 1, 17)} minDate={new Date(2017, 4, 17)}
    maxDate={new Date(2018, 5, 17)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
      <ViewDirective option='Agenda' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Agenda, Month]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;

```

```

        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

INDEX.CSS

```

.templatewrap {
    text-align: center;
    width: 100%;
}

.templatewrap img {
    width: 20px;
    height: 20px;
}

/* csslint ignore:start */
.schedule-cell-dimension.e-schedule .e-vertical-view .e-date-header-wrap
table col,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-content-wrap table
col {
    width: 200px;
}

.schedule-cell-dimension.e-schedule .e-vertical-view .e-time-cells-wrap
table td,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-work-cells {
    height: 100px;
}

.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells,
.schedule-cell-dimension.e-schedule .e-month-view .e-date-header-wrap table
col {
    width: 200px;
}

.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells {
    height: 200px;
}
/* csslint ignore:end */

```

By default, the `minDate` property value is set to `new Date(1900, 0, 1)` and `maxDate` property value is set to `new Date(2099, 11, 31)`. The user can also set the customized `minDate` and `maxDate` property values.

Customizing the weekend cells background color

You can customize the background color of weekend cells by utilizing the [renderCell](#) event and checking the [elementType](#) option within the event.

```
`ts
const onRendercell = (args) => {
  if (args.elementType == "workCells") {
    // To change the color of weekend columns in week view
    if (args.date) {
      if (args.date.getDay() === 6) {
        (args.element).style.background = '#ffdea2';
      }
      if (args.date.getDay() === 0) {
        (args.element).style.background = '#ffdea2';
      }
    }
  }
}
```

And, the background color for weekend cells in the Month view through the [cssClass](#) property, which overrides the default CSS applied on cells.

```
`css
.schedule-cell-customization.e-schedule .e-month-view .e-work-cells:not(.e-work-days) {
  background-color: #f08080;
}
```

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  const onRendercell = (args) => {
    if (args.elementType == "workCells") {
      // To change the color of weekend columns in week view
      if (args.date) {
        if (args.date.getDay() === 6) {
          (args.element).style.background = '#ffdea2';
        }
      }
    }
  }
}
```

```

        if (args.date.getDay() === 0) {
            (args.element).style.background = '#ffdea2';
        }
    }
}
};
return <ScheduleComponent cssClass="schedule-cell-customization"
width='100%' height='550px' selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings} renderCell={onRendercell}>
    <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
</ScheduleComponent>;
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
    ActionEventArgs, ViewsDirective, ViewDirective, RenderCellEventArgs
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onRendercell = (args: RenderCellEventArgs): void => {
        if (args.elementType === "workCells") {
            // To change the color of weekend columns in week view
            if (args.date) {
                if (args.date.getDay() === 6) {
                    (args.element).style.background = '#ffdea2';
                }
                if (args.date.getDay() === 0) {
                    (args.element).style.background = '#ffdea2';
                }
            }
        }
    };
    return <ScheduleComponent cssClass="schedule-cell-customization"
width='100%' height='550px' selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings} renderCell={onRendercell}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>

```

```
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```


How to disable multiple cell and row selection in Schedule

By default, the `allowMultiCellSelection` and `allowMultiRowSelection` properties of the Schedule are set to `true`. So, the Schedule allows user to select multiple cells and rows. If the user want to disable this multiple cell and row selection. The user can disable this feature by setting up `false` to these properties.

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

State persistence in React Schedule component

State persistence allowed Scheduler to retain the `currentView`, `selectedDate` and Scroll position values in the `localStorage` for state maintenance even if the browser is refreshed or if you move to the next page within the browser. This action is handled through the `enablePersistence` property which is set to `false` by default. When it is set to `true`, `currentView`, `selectedDate` and Scroll position values of the scheduler component will be retained even after refreshing the page.

Note: Scheduler `id` is essential to set state persistence.

The following sample demonstrates how to set state persistence of the Scheduler component.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject } from
 '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
    Date(2018, 1, 15)} eventSettings={eventSettings} enablePersistence={true}>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>;
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return <ScheduleComponent width='100%' height='550px' selectedDate={new
    Date(2018, 1, 15)} eventSettings={eventSettings} enablePersistence={true}>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Exporting in React Schedule component

The Scheduler supports exporting all its appointments both to an Excel or ICS extension file at client-side. It offers different client-side methods to export its appointments in an Excel or iCal format file. Let's look onto the ways on how to implement the exporting functionality in Scheduler.

Excel Exporting

The Scheduler allows you to export all its events into an Excel format file by using the `exportToExcel` client-side method. By default, it exports all the default fields of Scheduler mapped through `eventSettings` property.

Before you start with excel exporting functionality, you need to import and inject the `ExcelExport` module from the '@syncfusion/ej2-schedule' package using the `Inject` method of Scheduler.

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import { useRef } from 'react';
import * as React from 'react';
import { ScheduleComponent, ViewDirective, Week, Resize, ExcelExport,
DragAndDrop, Inject, ViewsDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onActionBegin = (args) => {
    if (args.requestType === 'toolbarItemRendering') {
      let exportItem = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-
schedule-excel-export',
        text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
      };
      args.items.push(exportItem);
    }
  }
  const onExportClick = () => {
    scheduleObj.current.exportToExcel();
  }
  return (
    <ScheduleComponent cssClass='excel-export' width='100%'
height='550px' id='schedule' ref={scheduleObj} selectedDate={new Date(2019,
0, 10)} eventSettings={eventSettings} actionBegin={onActionBegin}>
      <ViewsDirective>
        <ViewDirective option='Week' />
      </ViewsDirective>
      <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import { useRef } from 'react';
import * as React from 'react';
import { ItemModel } from '@syncfusion/ej2-react-navigations';
import {
    ScheduleComponent, ViewDirective, Week, Resize, ExcelExport, ExportOptions,
    ActionEventArgs, ToolbarActionArgs, DragAndDrop, Inject, ViewsDirective,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onActionBegin = (args: ActionEventArgs & ToolbarActionArgs): void => {
        if (args.requestType === 'toolbarItemRendering') {
            let exportItem: ItemModel = {
                align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
                excel-export',
                text: 'Excel Export', cssClass: 'e-excel-export', click:
                onExportClick
            };
            args.items.push(exportItem);
        }
    }
    const onExportClick = (): void => {
        scheduleObj.current.exportToExcel();
    }
    return (
        <ScheduleComponent cssClass='excel-export' width='100%' height='550px'
        id='schedule' ref={scheduleObj}
            selectedDate={new Date(2019, 0, 10)} eventSettings={eventSettings}
            actionBegin={onActionBegin}>
            <ViewsDirective>
                <ViewDirective option='Week' />
            </ViewsDirective>
            <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
        </ScheduleComponent>
    );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Exporting with custom fields

By default, Scheduler exports all the default event fields that are mapped to it through the `eventSettings` property. To limit the number of fields on the exported excel file, it provides an option to export only the custom fields of the event data. To export such custom fields alone, define the required fields through the `ExportOptions` interface and pass it as argument to the `exportToExcel` method as shown in the following example. For example: `['Id', 'Subject', 'StartTime', 'EndTime', 'Location']`.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';

```

```

import { ScheduleComponent, ViewDirective, Week, Resize, ExcelExport,
DragAndDrop, Inject, ViewsDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onActionBegin = (args) => {
    if (args.requestType === 'toolbarItemRendering') {
      let exportItem = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-
schedule-excel-export',
        text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
      };
      args.items.push(exportItem);
    }
  }
  const onExportClick = () => {
    let exportValues = {
      fields: ['Id', 'Subject', 'StartTime', 'EndTime', 'Location']
    };
    scheduleObj.current.exportToExcel(exportValues);
  }
  return (<ScheduleComponent cssClass='excel-export' width='100%'
height='550px' id='schedule' ref={scheduleObj} selectedDate={new Date(2019,
0, 10)} eventSettings={eventSettings} actionBegin={onActionBegin}>
    <ViewsDirective>
      <ViewDirective option='Week' />
    </ViewsDirective>
    <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ItemModel } from '@syncfusion/ej2-react-navigations';
import {
  ScheduleComponent, ViewDirective, Week, Resize, ExcelExport, ExportOptions,
  ActionEventArgs, ToolbarEventArgs, DragAndDrop, Inject, ViewsDirective,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);

```

```

const eventSettings: EventSettingsModel = { dataSource: scheduleData };
const onActionBegin = (args: ActionEventArgs & ToolbarActionArgs): void =>
{
    if (args.requestType === 'toolbarItemRendering') {
        let exportItem: ItemModel = {
            align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
            text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
        };
        args.items.push(exportItem);
    }
}
const onExportClick = (): void => {
    let exportValues: ExportOptions = {
        fields: ['Id', 'Subject', 'StartTime', 'EndTime', 'Location']
    };
    scheduleObj.current.exportToExcel(exportValues);
}
return (
    <ScheduleComponent cssClass='excel-export' width='100%' height='550px'
id='schedule' ref={scheduleObj}
        selectedDate={new Date(2019, 0, 10)} eventSettings={eventSettings}
        actionBegin={onActionBegin}>
        <ViewsDirective>
            <ViewDirective option='Week' />
        </ViewsDirective>
        <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
    </ScheduleComponent>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Exporting individual occurrences of a recurring series

By default, the Scheduler exports recurring events as a single data by exporting only its parent record into the excel file. If you want to export each individual occurrences of a recurring series appointment as separate records in an Excel file, define the `includeOccurrences` option as `true` through the `ExportOptions` interface and pass it as argument to the `exportToExcel` method. By default, the `includeOccurrences` option is set to `false`.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, ViewDirective, Week, Resize, ExcelExport,
DragAndDrop, Inject, ViewsDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
    const scheduleObj = useRef(null);
    const eventSettings = { dataSource: scheduleData };
    const onActionBegin = (args) => {
        if (args.requestType === 'toolbarItemRendering') {
            let exportItem = {

```



```

        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-
schedule-excel-export',
        text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
    };
    args.items.push(exportItem);
}
}
const onExportClick = () => {
    let exportValues = { includeOccurrences: true };
    scheduleObj.current.exportToExcel(exportValues);
}
return (<ScheduleComponent cssClass='excel-export' width='100%'
height='550px' id='schedule' ref={scheduleObj} selectedDate={new Date(2019,
0, 10)} eventSettings={eventSettings} actionBegin={onActionBegin}>
    <ViewsDirective>
        <ViewDirective option='Week' />
    </ViewsDirective>
    <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ItemModel } from '@syncfusion/ej2-react-navigations';
import {
    ScheduleComponent, ViewDirective, Week, Resize, ExcelExport, ExportOptions,
    ActionEventArgs, ToolbarActionArgs, DragAndDrop, Inject, ViewsDirective,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onActionBegin = (args: ActionEventArgs & ToolbarActionArgs): void =>
    {
        if (args.requestType === 'toolbarItemRendering') {
            let exportItem: ItemModel = {
                align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
                text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
            };
            args.items.push(exportItem);
        }
    }
    const onExportClick = (): void => {

```

```

    let exportValues: ExportOptions = { includeOccurrences: true };
    scheduleObj.current.exportToExcel (exportValues);
  }
  return (
    <ScheduleComponent cssClass='excel-export' width='100%' height='550px'
    id='schedule' ref={scheduleObj}
      selectedDate={new Date(2019, 0, 10)} eventSettings={eventSettings}
      actionBegin={onActionBegin}>
      <ViewsDirective>
        <ViewDirective option='Week' />
      </ViewsDirective>
      <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot (document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdn.jsdelivr.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
    }
  </style>

```

```

        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Exporting custom event data

By default, the whole event collection bound to the Scheduler gets exported as an excel file. To export only specific events of Scheduler or some custom event collection, you need to pass those custom data collection as a parameter to the `exportToExcel` method as shown in this following example, through the `customData` option of `ExportOptions` interface.

By default, the event data are taken from Scheduler `dataSource`.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, ViewDirective, Week, Resize, ExcelExport,
DragAndDrop, Inject, ViewsDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
    const scheduleObj = useRef(null);
    const eventSettings = { dataSource: scheduleData };
    const onActionBegin = (args) => {
        if (args.requestType === 'toolbarItemRendering') {
            let exportItem = {
                align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-
schedule-excel-export',
                text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
            };
            args.items.push(exportItem);
        }
    }
    const onExportClick = () => {
        let exportValues = {
            customData: [{
                Id: 1,
                Subject: 'Explosion of Betelgeuse Star',
                Location: 'Space Centre USA',
                StartTime: new Date(2019, 0, 6, 9, 30),
                EndTime: new Date(2019, 0, 6, 11, 0),
                CategoryColor: '#1aaa55'
            }

```

```

    }, {
      Id: 2,
      Subject: 'Thule Air Crash Report',
      Location: 'Newyork City',
      StartTime: new Date(2019, 0, 7, 12, 0),
      EndTime: new Date(2019, 0, 7, 14, 0),
      CategoryColor: '#357cd2'
    }]
  };
  scheduleObj.current.exportToExcel (exportValues);
}
return (<ScheduleComponent cssClass='excel-export' width='100%'
height='550px' id='schedule' ref={scheduleObj} selectedDate={new Date(2019,
0, 10)} eventSettings={eventSettings} actionBegin={onActionBegin}>
  <ViewsDirective>
    <ViewDirective option='Week' />
  </ViewsDirective>
  <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot (document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ItemModel } from '@syncfusion/ej2-react-navigations';
import {
  ScheduleComponent, ViewDirective, Week, Resize, ExcelExport, ExportOptions,
  ActionEventArgs, ToolbarActionArgs, DragAndDrop, Inject, ViewsDirective,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const onActionBegin = (args: ActionEventArgs & ToolbarActionArgs): void =>
  {
    if (args.requestType === 'toolbarItemRendering') {
      let exportItem: ItemModel = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
        text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
      };
      args.items.push(exportItem);
    }
  }
  const onExportClick = (): void => {
    let exportValues: ExportOptions = {

```

```

        customData: [{
            Id: 1,
            Subject: 'Explosion of Betelgeuse Star',
            Location: 'Space Centre USA',
            StartTime: new Date(2019, 0, 6, 9, 30),
            EndTime: new Date(2019, 0, 6, 11, 0),
            CategoryColor: '#1aaa55'
        }, {
            Id: 2,
            Subject: 'Thule Air Crash Report',
            Location: 'Newyork City',
            StartTime: new Date(2019, 0, 7, 12, 0),
            EndTime: new Date(2019, 0, 7, 14, 0),
            CategoryColor: '#357cd2'
        }]
    };
    scheduleObj.current.exportToExcel (exportValues);
}
return (
    <ScheduleComponent cssClass='excel-export' width='100%' height='550px'
    id='schedule' ref={scheduleObj}
        selectedDate={new Date(2019, 0, 10)} eventSettings={eventSettings}
        actionBegin={onActionBegin}>
        <ViewsDirective>
            <ViewDirective option='Week' />
        </ViewsDirective>
        <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
    </ScheduleComponent>
);
};
const root = ReactDOM.createRoot (document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Customizing column header with custom fields exporting

Using fields property, we can only export the defined fields into excel without customizing the header. Now we can provide the alternate support to customize the header of custom fields exporting using the fieldsInfo option through the ExportFieldInfo interface and pass it as an argument to the exportToExcel method as shown in the following example.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, ViewDirective, Week, Resize, ExcelExport,
DragAndDrop, Inject, ViewsDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
    const scheduleObj = useRef(null);
    const eventSettings = { dataSource: scheduleData };
    const onActionBegin = (args) => {
        if (args.requestType === 'toolbarItemRendering') {
            let exportItem = {
                align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-
schedule-excel-export',

```

```

        text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
    };
    args.items.push(exportItem);
}
}
const onExportClick = () => {
    let customFields = [
        { name: 'Subject', text: 'Summary' },
        { name: 'StartTime', text: 'First Date' },
        { name: 'EndTime', text: 'Last Date' },
        { name: 'Location', text: 'Place' },
        { name: 'OwnerId', text: 'Owners' }
    ];
    let exportValues = { fieldsInfo: customFields };
    scheduleObj.current.exportToExcel(exportValues);
}
return (<ScheduleComponent cssClass='excel-export' width='100%'
height='550px' id='schedule' ref={scheduleObj} selectedDate={new Date(2019,
0, 10)} eventSettings={eventSettings} actionBegin={onActionBegin}>
    <ViewsDirective>
        <ViewDirective option='Week' />
    </ViewsDirective>
    <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ItemModel } from '@syncfusion/ej2-react-navigations';
import {
    ScheduleComponent, ViewDirective, Week, Resize, ExcelExport, ExportOptions,
    ExportFieldInfo,
    ActionEventArgs, ToolbarActionArgs, DragAndDrop, Inject, ViewsDirective,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onActionBegin = (args: ActionEventArgs & ToolbarActionArgs): void =>
    {
        if (args.requestType === 'toolbarItemRendering') {
            let exportItem: ItemModel = {
                align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',

```

```

        text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
    };
    args.items.push(exportItem);
}
}
const onExportClick = (): void => {
    let customFields: ExportFieldInfo[] = [
        { name: 'Subject', text: 'Summary' },
        { name: 'StartTime', text: 'First Date' },
        { name: 'EndTime', text: 'Last Date' },
        { name: 'Location', text: 'Place' },
        { name: 'OwnerId', text: 'Owners' }
    ];
    let exportValues: ExportOptions = { fieldsInfo: customFields };
    scheduleObj.current.exportToExcel(exportValues);
}
return (
    <ScheduleComponent cssClass='excel-export' width='100%' height='550px'
id='schedule' ref={scheduleObj}
        selectedDate={new Date(2019, 0, 10)} eventSettings={eventSettings}
        actionBegin={onActionBegin}>
        <ViewsDirective>
            <ViewDirective option='Week' />
        </ViewsDirective>
        <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
    </ScheduleComponent>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />

```



```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008c9f;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Export with custom file name

By default, the Scheduler allows you to download the exported Excel file with a name `Schedule.xlsx`. It also provides an option to export the excel file with a custom file name, by defining the desired `fileName` through the `ExportOptions` interface and passing it as an argument to the `exportToExcel` method.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, ViewDirective, Week, Resize, ExcelExport,
DragAndDrop, Inject, ViewsDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
    const scheduleObj = useRef(null);
    const eventSettings = { dataSource: scheduleData };
    const onActionBegin = (args) => {
        if (args.requestType === 'toolbarItemRendering') {
            let exportItem = {
                align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-
schedule-excel-export',
                text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick

```

```

        };
        args.items.push(exportItem);
    }
}
const onExportClick = () => {
    let exportValues = { fileName: "SchedulerData" };
    scheduleObj.current.exportToExcel(exportValues);
}
return (<ScheduleComponent cssClass='excel-export' width='100%'
height='550px' id='schedule' ref={scheduleObj} selectedDate={new Date(2019,
0, 10)} eventSettings={eventSettings} actionBegin={onActionBegin}>
    <ViewsDirective>
        <ViewDirective option='Week' />
    </ViewsDirective>
    <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ItemModel } from '@syncfusion/ej2-react-navigations';
import {
    ScheduleComponent, ViewDirective, Week, Resize, ExcelExport, ExportOptions,
    ActionEventArgs, ToolbarActionArgs, DragAndDrop, Inject, ViewsDirective,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onActionBegin = (args: ActionEventArgs & ToolbarActionArgs): void =>
    {
        if (args.requestType === 'toolbarItemRendering') {
            let exportItem: ItemModel = {
                align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
                text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
            };
            args.items.push(exportItem);
        }
    }
    const onExportClick = (): void => {
        let exportValues: ExportOptions = { fileName: "SchedulerData" };
        scheduleObj.current.exportToExcel(exportValues);
    }
    return (

```

```

<ScheduleComponent cssClass='excel-export' width='100%' height='550px'
id='schedule' ref={scheduleObj}
    selectedDate={new Date(2019, 0, 10)} eventSettings={eventSettings}
    actionBegin={onActionBegin} >
    <ViewsDirective>
        <ViewDirective option='Week' />
    </ViewsDirective>
    <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
</ScheduleComponent >
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <link href="index.css" rel="stylesheet" />
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;

```

```

    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Excel file formats

By default, the Scheduler exports event data to an excel file in the .xlsx format. You can also export the Scheduler data in either of the file type such as .xlsx or csv formats, by defining the exportType option as either csv or xlsx through the ExportOptions interface. By default, the exportType is set to xlsx.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, ViewDirective, Week, Resize, ExcelExport,
DragAndDrop, Inject, ViewsDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onActionBegin = (args) => {
    if (args.requestType === 'toolbarItemRendering') {
      let exportItem = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-
schedule-excel-export',
        text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
      };
      args.items.push(exportItem);
    }
  }
  const onExportClick = () => {
    let exportValues = { exportType: "csv" };
    scheduleObj.current.exportToExcel(exportValues);
  }
  return (<ScheduleComponent cssClass='excel-export' width='100%'
height='550px' id='schedule' ref={t => scheduleObj = t} selectedDate={new
Date(2019, 0, 10)} eventSettings={eventSettings} actionBegin={onActionBegin}>
    <ViewsDirective>
      <ViewDirective option='Week' />
    </ViewsDirective>
    <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ItemModel } from '@syncfusion/ej2-react-navigations';
import {
    ScheduleComponent, ViewDirective, Week, Resize, ExcelExport, ExportOptions,
    ActionEventArgs, ToolbarActionArgs, DragAndDrop, Inject, ViewsDirective,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onActionBegin = (args: ActionEventArgs & ToolbarActionArgs): void =>
    {
        if (args.requestType === 'toolbarItemRendering') {
            let exportItem: ItemModel = {
                align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
                text: 'Excel Export', cssClass: 'e-excel-export', click:
onExportClick
            };
            args.items.push(exportItem);
        }
    }
    const onExportClick = (): void => {
        let exportValues: ExportOptions = { exportType: "csv" };
        scheduleObj.current.exportToExcel(exportValues);
    }
    return (
        <ScheduleComponent cssClass='excel-export' width='100%' height='550px'
id='schedule' ref={scheduleObj}
        selectedDate={new Date(2019, 0, 10)} eventSettings={eventSettings}
        actionBegin={onActionBegin}>
            <ViewsDirective>
                <ViewDirective option='Week' />
            </ViewsDirective>
            <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
        </ScheduleComponent>
    );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>

```

```

<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta name="description" content="Essential JS 2 for React Components" />
<meta name="author" content="Syncfusion" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Custom separator in CSV

The Scheduler exports the event data to CSV format with `,` as separator. You can change separator by setting [separator](#) property in [ExportOptions](#).

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';

```

```

import { ScheduleComponent, ViewDirective, Week, Resize, ExcelExport,
DragAndDrop, Inject, ViewsDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule Export Custom Separator
 */
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onActionBegin = (args) => {
    if (args.requestType === 'toolbarItemRendering') {
      let exportItem = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-
schedule-excel-export',
        text: 'CSV-Export', cssClass: 'e-excel-export', click:
onExportClick
      };
      args.items.push(exportItem);
    }
  }
  const onExportClick = () => {
    let exportValues = { exportType: 'csv', separator: ';' };
    scheduleObj.current.exportToExcel(exportValues);
  }
  return (<ScheduleComponent cssClass='excel-export' width='100%'
height='550px' id='schedule' ref={scheduleObj} selectedDate={new Date(2019,
0, 10)} eventSettings={eventSettings} actionBegin={onActionBegin}>
    <ViewsDirective>
      <ViewDirective option='Week' />
    </ViewsDirective>
    <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
  </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ItemModel } from '@syncfusion/ej2-react-navigations';
import {
  ScheduleComponent, ViewDirective, Week, Resize, ExcelExport, ExportOptions,
  ActionEventArgs, ToolbarEventArgs, DragAndDrop, Inject, ViewsDirective,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
/**
 * Schedule Export Custom Separator
 */
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };

```

```

const onActionBegin = (args: ActionEventArgs & ToolbarActionArgs): void =>
{
    if (args.requestType === 'toolbarItemRendering') {
        let exportItem: ItemModel = {
            align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
            text: 'CSV-Export', cssClass: 'e-excel-export', click: onExportClick
        };
        args.items.push(exportItem);
    }
}

const onExportClick = (): void => {
    let exportValues: ExportOptions = { exportType: 'csv', separator: ';' };
    scheduleObj.current.exportToExcel(exportValues);
}

return (
    <ScheduleComponent cssClass='excel-export' width='100%' height='550px'
id='schedule' ref={scheduleObj}
    selectedDate={new Date(2019, 0, 10)} eventSettings={eventSettings}
    actionBegin={onActionBegin}>
        <ViewsDirective>
            <ViewDirective option='Week' />
        </ViewsDirective>
        <Inject services={[Week, Resize, DragAndDrop, ExcelExport]} />
    </ScheduleComponent>
);
};

const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />

```



```

    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <link href="index.css" rel="stylesheet" />
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008c9f;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Exporting calendar events as ICS file

You can export the Scheduler events to a calendar (.ics) file format, and open it on any of the other default calendars such as Google or Outlook. To export the events of Scheduler to an ICS file, you need to first import the `ICalendarExport` module from `@syncfusion/ej2-schedule` package and then inject it using the `Schedule.Inject(ICalendarExport)` method.

The following code example shows how the Scheduler events are exported to a calendar (.ics) file by making use of the `exportToICalendar` public method.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
    ICalendarExport, ICalendarImport, Resize, Inject
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
    const scheduleObj = useRef(null);
    const eventSettings = { dataSource: scheduleData };
    const onClick = () => {
        scheduleObj.current.exportToICalendar();
    }
}

```

```

return (<div>
  <ButtonComponent id='ics-export' title='Export'
onClick={onClick}>Export</ButtonComponent>
  <ScheduleComponent ref={scheduleObj} width='100%' height='520px'
id='schedule' selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
ICalendarExport, ICalendarImport, Resize, DragAndDrop]} />
  </ScheduleComponent></div>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
  ICalendarExport, ICalendarImport, Resize, Inject
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings = { dataSource: scheduleData };
  const onClick = (): void => {
    scheduleObj.current.exportToICalendar();
  }
  return (<div>
    <ButtonComponent id='ics-export' title='Export'
onClick={onClick}>Export</ButtonComponent>
    <ScheduleComponent ref={scheduleObj} width='100%' height='520px'
id='schedule' selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
ICalendarExport, ICalendarImport, Resize, DragAndDrop]} />
    </ScheduleComponent></div>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

```

```

    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <link href="index.css" rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

INDEX.CSS

```

#container {
    visibility: hidden;
}

#loader {
    color: #008cff;
    height: 40px;
    width: 30%;
    position: absolute;
    top: 45%;
}

```

```

    left: 45%;
  }

  .e-schedule .e-schedule-toolbar .user-icon,
  .e-profile-wrapper .profile-image {
    background-image:
url('https://ej2.syncfusion.com/demos/src/schedule/images/nancy.png');
    background-position: center center;
    background-repeat: no-repeat;
    background-size: cover;
    border-radius: 50%;
  }

  /* csslint ignore:start */
  .e-schedule .e-schedule-toolbar .user-icon {
    height: 24px;
    min-width: 24px !important;
    width: 24px !important;
  }
  /* csslint ignore:end */

  .e-schedule .e-schedule-toolbar .e-toolbar-items .e-schedule-user-icon .e-
tbar-btn:hover {
    background-color: inherit;
  }

  .e-schedule .e-schedule-toolbar .e-toolbar-items .e-schedule-user-icon .e-
tbar-btn-text {
    display: none;
  }

  /* csslint ignore:start */
  .e-schedule .e-schedule-toolbar .e-toolbar-pop .e-schedule-user-icon .e-
tbar-btn-text {
    padding-left: 8px !important;
  }
  /* csslint ignore:end */

  .e-profile-wrapper {
    width: 210px;
    height: 80px;
    background-color: #fafafa;
    box-shadow: inset 0 0 5px rgba(0, 0, 0, 0.2);
    overflow: hidden;
  }

  .e-profile-wrapper .profile-container {
    display: flex;
    padding: 10px;
  }

  .e-profile-wrapper .profile-image {
    box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0, 0, 0, 0.2);
    width: 60px;
    height: 60px;
  }

```

```
.e-profile-wrapper .content-wrap {
  padding-left: 10px;
}

.e-profile-wrapper .name {
  font-size: 14px;
  line-height: 20px;
  font-weight: 500;
  margin-top: 2px;
}

.e-profile-wrapper .destination {
  font-size: 12px;
}

.e-profile-wrapper .status-icon {
  height: 6px;
  width: 6px;
  background: green;
  border-radius: 100%;
  float: left;
  margin-right: 4px;
  margin-top: 4px;
}

.e-profile-wrapper .status {
  font-size: 11px;
}
```

Exporting calendar with custom file name

By default, the calendar is exported with a file name `Calendar.ics`. To change this file name on export, pass the custom string value as `fileName` to the method argument so as to get the file downloaded with this provided name.

The following example downloads the iCal file with a name `ScheduleEvents.ics`.

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
ICalendarExport, ICalendarImport, Resize, Inject } from '@syncfusion/ej2-
react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onClick = () => {
    scheduleObj.current.exportToICalendar('ScheduleEvents');
  }
}
```

```

    return (<div>
      <ButtonComponent id='ics-export' title='Export'
onClick={onClick}>Export</ButtonComponent>
      <ScheduleComponent ref={scheduleObj} width='100%' height='520px'
id='schedule' selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
ICalendarExport, ICalendarImport, Resize, DragAndDrop]} />
      </ScheduleComponent></div>);
  }
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
  ICalendarExport, ICalendarImport, Resize, Inject
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { scheduleData } from './datasource';
/**
 * Schedule header customization sample
 */
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings = { dataSource: scheduleData };
  const onClick = (): void => {
    scheduleObj.current.exportToICalendar('ScheduleEvents');
  }
  return (<div>
    <ButtonComponent id='ics-export' title='Export'
onClick={onClick}>Export</ButtonComponent>
    <ScheduleComponent ref={scheduleObj} width='100%' height='520px'
id='schedule' selectedDate={new Date(2018, 1, 15)}
eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
ICalendarExport, ICalendarImport, Resize, DragAndDrop]} />
    </ScheduleComponent></div>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

```

```

<meta name="description" content="Essential JS 2 for React Components" />
<meta name="author" content="Syncfusion" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

INDEX.CSS

```

#container {
    visibility: hidden;
}

#loader {
    color: #008cff;
    height: 40px;
    width: 30%;
    position: absolute;
    top: 45%;
}

```

```
    left: 45%;
  }

  .e-schedule .e-schedule-toolbar .user-icon,
  .e-profile-wrapper .profile-image {
    background-image:
url('https://ej2.syncfusion.com/demos/src/schedule/images/nancy.png');
    background-position: center center;
    background-repeat: no-repeat;
    background-size: cover;
    border-radius: 50%;
  }

  /* csslint ignore:start */
  .e-schedule .e-schedule-toolbar .user-icon {
    height: 24px;
    min-width: 24px !important;
    width: 24px !important;
  }
  /* csslint ignore:end */

  .e-schedule .e-schedule-toolbar .e-toolbar-items .e-schedule-user-icon .e-
tbar-btn:hover {
    background-color: inherit;
  }

  .e-schedule .e-schedule-toolbar .e-toolbar-items .e-schedule-user-icon .e-
tbar-btn-text {
    display: none;
  }

  /* csslint ignore:start */
  .e-schedule .e-schedule-toolbar .e-toolbar-pop .e-schedule-user-icon .e-
tbar-btn-text {
    padding-left: 8px !important;
  }
  /* csslint ignore:end */

  .e-profile-wrapper {
    width: 210px;
    height: 80px;
    background-color: #fafafa;
    box-shadow: inset 0 0 5px rgba(0, 0, 0, 0.2);
    overflow: hidden;
  }

  .e-profile-wrapper .profile-container {
    display: flex;
    padding: 10px;
  }

  .e-profile-wrapper .profile-image {
    box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0, 0, 0, 0.2);
    width: 60px;
    height: 60px;
  }
}
```



```

.e-profile-wrapper .content-wrap {
  padding-left: 10px;
}

.e-profile-wrapper .name {
  font-size: 14px;
  line-height: 20px;
  font-weight: 500;
  margin-top: 2px;
}

.e-profile-wrapper .destination {
  font-size: 12px;
}

.e-profile-wrapper .status-icon {
  height: 6px;
  width: 6px;
  background: green;
  border-radius: 100%;
  float: left;
  margin-right: 4px;
  margin-top: 4px;
}

.e-profile-wrapper .status {
  font-size: 11px;
}

```

Import events from other calendars

The events from external calendars (ICS files) can be imported into Scheduler by using the `importICalendar` method. This method accepts the `blob object` of an .ics file to be imported, as a mandatory argument.

To import an ICS file containing events into Scheduler, you need to first import the `ICalendarImport` module from `@syncfusion/ej2-schedule` package and then inject it using the `Schedule.Inject(ICalendarImport)` method.

The following example shows how to import an ICS file into Scheduler, using the `importICalendar` method.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Resize,
  Inject, ICalendarExport, ICalendarImport
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
import { UploaderComponent } from '@syncfusion/ej2-react-inputs';
const App = () => {
  const scheduleObj = useRef(null);

```

```

let multiple = false;
let showFileList = false;
const allowedExtensions = '.ics';
const eventSettings = { dataSource: scheduleData };
const onSelect = (args) => {
  scheduleObj.current.importICalendar(args.event.target.files[0]);
}
return (<div>
  <UploaderComponent id='fileUpload' type='file'
allowedExtensions={allowedExtensions} cssClass='calendar-import'
  buttons={{ browse: 'Choose file' }} multiple={multiple}
showFileList={showFileList}
  selected={onSelect}></UploaderComponent>
  <ScheduleComponent ref={scheduleObj} width='100%' height='520px'
selectedDate={new Date(2018, 1, 15)} allowDragAndDrop={false}
eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
ICalendarExport, ICalendarImport, Resize, DragAndDrop]} />
  </ScheduleComponent></div>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Resize,
  Inject, ICalendarExport, ICalendarImport
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
import { UploaderComponent } from '@syncfusion/ej2-react-inputs';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  let multiple: boolean = false;
  let showFileList: boolean = false;
  const allowedExtensions: string = '.ics';
  const eventSettings = { dataSource: scheduleData };
  const onSelect = (args): void => {
    scheduleObj.current.importICalendar(args.event.target.files[0]);
  }
  return (<div>
    <UploaderComponent id='fileUpload' type='file'
allowedExtensions={allowedExtensions} cssClass='calendar-import'
  buttons={{ browse: 'Choose file' }} multiple={multiple}
showFileList={showFileList}
  selected={onSelect}></UploaderComponent>
    <ScheduleComponent ref={scheduleObj} width='100%' height='520px'
selectedDate={new Date(2018, 1, 15)} allowDragAndDrop={false}
eventSettings={eventSettings}>
      <Inject services={[Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
ICalendarExport, ICalendarImport, Resize, DragAndDrop]} />
    </ScheduleComponent></div>

```

```
);  
};  
const root = ReactDOM.createRoot(document.getElementById('schedule'));  
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <title>Syncfusion React Schedule</title>  
  <meta charset="utf-8" />  
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
  <meta name="description" content="Essential JS 2 for React Components" />  
  <meta name="author" content="Syncfusion" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-buttons/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-calendars/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-dropdowns/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-inputs/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-navigations/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-popups/styles/material.css" rel="stylesheet" />  
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-schedule/styles/material.css" rel="stylesheet" />  
  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />  
  <link href="index.css" rel="stylesheet" />  
  <script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></script>  
  <script src="systemjs.config.js"></script>  
  <style>  
    #loader {  
      color: #008cff;  
      height: 40px;  
      left: 45%;  
      position: absolute;  
      top: 45%;  
      width: 30%;  
    }  
  </style>  
</head>  
<body>  
  <div id='schedule'>  
    <div id='loader'>Loading....</div>  
  </div>  
</body>  
</html>
```

INDEX.CSS

```
/* csslint ignore:start */
#container {
  visibility: hidden;
}

#loader {
  color: #008cff;
  height: 40px;
  width: 30%;
  position: absolute;
  top: 45%;
  left: 45%;
}

.calendar-import.e-upload {
  border: 0;
  padding-left: 0 !important;
}

.calendar-import.e-upload .e-file-select-wrap {
  padding: 0
}

.calendar-import.e-upload .e-file-select-wrap .e-file-drop {
  display: none;
}

/* csslint ignore:end */

.e-profile-wrapper {
  width: 210px;
  height: 80px;
  background-color: #fafafa;
  box-shadow: inset 0 0 5px rgba(0, 0, 0, 0.2);
  overflow: hidden;
}

.e-profile-wrapper .profile-container {
  display: flex;
  padding: 10px;
}

.e-profile-wrapper .profile-image {
  box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0, 0, 0, 0.2);
  width: 60px;
  height: 60px;
}

.e-profile-wrapper .content-wrap {
  padding-left: 10px;
}

.e-profile-wrapper .name {
```

```

    font-size: 14px;
    line-height: 20px;
    font-weight: 500;
    margin-top: 2px;
  }

  .e-profile-wrapper .destination {
    font-size: 12px;
  }

  .e-profile-wrapper .status-icon {
    height: 6px;
    width: 6px;
    background: green;
    border-radius: 100%;
    float: left;
    margin-right: 4px;
    margin-top: 4px;
  }

  .e-profile-wrapper .status {
    font-size: 11px;
  }
/* csslint ignore:end */

```

How to print the Scheduler element

The Scheduler allows you to print the Scheduler element by using the `print` client-side method. The `print` method works in two ways. You can find it below.

- Using `print` method without options.
- Using a `print` method with options.

To print the Schedule, you need to import the `Print` module from the `@syncfusion/ej2-react-schedule` package and then inject it using `<Inject services={[Print]} />`.

Using `print` method without options

You can print the Schedule element with the current view by using the `print` method without passing the options. The following example shows how to print the Scheduler using the `print` method without passing options.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Print, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onActionBegin = (args) => {
    if (args.requestType === 'toolbarItemRendering') {
      let exportItem = {

```

```

        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
print',
        text: 'Print', cssClass: 'e-schedule-print', click: onPrintIconClick
    };
    args.items.push(exportItem);
}
}
const onPrintIconClick = () => {
    scheduleObj.current.print();
}
return (<div>
    <ScheduleComponent ref={scheduleObj} width='100%' height='520px'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
actionBegin={onActionBegin}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda, Print]} />
    </ScheduleComponent></div>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda,
EventSettingsModel, Print, Inject, ActionEventArgs, ToolbarActionArgs } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
import { ItemModel } from '@syncfusion/ej2-react-navigations';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onActionBegin = (args: ActionEventArgs & ToolbarActionArgs): void =>
    {
        if (args.requestType === 'toolbarItemRendering') {
            let exportItem: ItemModel = {
                align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
print',
                text: 'Print', cssClass: 'e-schedule-print', click: onPrintIconClick
            };
            args.items.push(exportItem);
        }
    }
    const onPrintIconClick = (): void => {
        scheduleObj.current.print();
    }
    return (<div>
        <ScheduleComponent ref={scheduleObj} width='100%' height='520px'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
actionBegin={onActionBegin}>
            <Inject services={[Day, Week, WorkWeek, Month, Agenda, Print]} />
        </ScheduleComponent></div>
    );
};

```

```
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

Using a print method with options

You can print the Schedule element based on your needs using the `print` method by passing the print options used in this example with its values. The following example shows how to print the Scheduler using the `print` method by passing the options.

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Print,
Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onActionBegin = (args) => {
    if (args.requestType === 'toolbarItemRendering') {
      let exportItem = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
print',
        text: 'Print', cssClass: 'e-schedule-print', click: onPrintIconClick
      };
      args.items.push(exportItem);
    }
  }
  const onPrintIconClick = () => {
    let printModel = {
      agendaDaysCount: 14,
      cssClass: 'e-print-schedule',
      currentView: scheduleObj.current.currentView,
      dateFormat: 'dd-MMM-yyyy',
      enableRtl: false,
      endHour: '18:00',
      firstDayOfWeek: 1,
      firstMonthOfYear: 0,
      group: {},
      height: 'auto',
      locale: scheduleObj.current.locale,
      maxDate: scheduleObj.current.selectedDate,
      minDate: scheduleObj.current.getCurrentViewDates()[0],
      readonly: true,
      resources: [],
      rowAutoHeight: false,
      selectedDate: new Date(),
      showHeaderBar: false,
      showTimeIndicator: false,
      showWeekNumber: false,
      showWeekend: false,
      startHour: '06:00',
      timeFormat: 'HH',
      timeScale: { enable: true },
      width: 'auto',
      workDays: [1, 2, 3, 4, 5],
      workHours: { highlight: true, start: '10:00', end: '20:00' }
    };
  };
};
```



```

    scheduleObj.current.print(printModel);
  }
  return (
    <div>
      <ScheduleComponent ref={scheduleObj} width='100%' height='520px'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
actionBegin={onActionBegin}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda, Print]} />
      </ScheduleComponent></div>
    );
  };
  const root = ReactDOM.createRoot(document.getElementById('schedule'));
  root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Print,
EventSettingsModel, Inject, ScheduleModel, ActionEventArgs, ToolbarActionArgs
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
import { ItemModel } from '@syncfusion/ej2-react-navigations';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const onActionBegin = (args: ActionEventArgs & ToolbarActionArgs): void =>
  {
    if (args.requestType === 'toolbarItemRendering') {
      let exportItem: ItemModel = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
print',
        text: 'Print', cssClass: 'e-schedule-print', click: onPrintIconClick
      };
      args.items.push(exportItem);
    }
  }
  const onPrintIconClick = (): void => {
    let printModel: ScheduleModel = {
      agendaDaysCount: 14,
      cssClass: 'e-print-schedule',
      currentView: scheduleObj.current.currentView,
      dateFormat: 'dd-MMM-yyyy',
      enableRtl: false,
      endHour: '18:00',
      firstDayOfWeek: 1,
      firstMonthOfYear: 0,
      group: {},
      height: 'auto',
      locale: scheduleObj.current.locale,
      maxDate: scheduleObj.current.selectedDate,
      minDate: scheduleObj.current.getCurrentViewDates()[0],
      readonly: true,
      resources: [],
      rowAutoHeight: false,

```

```

        selectedDate: new Date(),
        showHeaderBar: false,
        showTimeIndicator: false,
        showWeekNumber: false,
        showWeekend: false,
        startHour: '06:00',
        timeFormat: 'HH',
        timeScale: { enable: true },
        width: 'auto',
        workDays: [1, 2, 3, 4, 5],
        workHours: { highlight: true, start: '10:00', end: '20:00' }
    };
    scheduleObj.current.print(printModel);
}
return (
    <div>
        <ScheduleComponent ref={scheduleObj} width='100%' height='520px'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
actionBegin={onActionBegin}>
            <Inject services={[Day, Week, WorkWeek, Month, Agenda, Print]} />
        </ScheduleComponent></div>
    );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <link href="index.css" rel="stylesheet" />

```

```

<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Context menu in React Schedule component

You can display context menu on work cells and appointments of Scheduler by making use of the **ContextMenu** control manually from the application end. In the following code example, context menu control is being added from sample end and set its target as **Scheduler**.

On Scheduler cells, you can display the menu items such as **New Event**, **New Recurring Event** and **Today** option. For appointments, you can display its related options such as **Edit Event** and **Delete Event**. The default event window can be opened for appointment creation and editing using the **openEditor** method of Scheduler.

The deletion of appointments can be done by using the **deleteEvent** public method. Also, the **selectedDate** property can be used to navigate between different dates.

You can also display custom menu options on Scheduler cells and appointments. Context menu will open on tap-hold in responsive mode.

INDEX.JSX

```

import { useRef, useState, useMemo, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { closest, isNullOrUndefined, remove, removeClass } from
'@syncfusion/ej2-base';
import { Query, DataManager } from '@syncfusion/ej2-data';
import {
  ScheduleComponent, ViewsDirective, ViewDirective,
  Day, Week, WorkWeek, Month, Agenda, Inject
} from '@syncfusion/ej2-react-schedule';
import { ContextMenuComponent } from '@syncfusion/ej2-react-navigations';
import { scheduleData } from '../datasource';

```

```

const App = () => {
  const scheduleObj = useRef(null);
  const menuObj = useRef(null);
  const [eventObj, setEventObj] = useState(null);
  const eventSettings = { dataSource: scheduleData };
  let selectedTarget;
  const menuItems = useMemo(() => [{
    text: 'New Event',
    iconCss: 'e-icons new',
    id: 'Add'
  }, {
    text: 'New Recurring Event',
    iconCss: 'e-icons recurrence',
    id: 'AddRecurrence'
  }, {
    text: 'Today',
    iconCss: 'e-icons today',
    id: 'Today'
  }, {
    text: 'Edit Event',
    iconCss: 'e-icons edit',
    id: 'Save'
  }, {
    text: 'Edit Event',
    id: 'EditRecurrenceEvent',
    iconCss: 'e-icons edit',
    items: [{
      text: 'Edit Occurrence',
      id: 'EditOccurrence'
    }, {
      text: 'Edit Series',
      id: 'EditSeries'
    }]
  }, {
    text: 'Delete Event',
    iconCss: 'e-icons delete',
    id: 'Delete'
  }, {
    text: 'Delete Event',
    id: 'DeleteRecurrenceEvent',
    iconCss: 'e-icons delete',
    items: [{
      text: 'Delete Occurrence',
      id: 'DeleteOccurrence'
    }, {
      text: 'Delete Series',
      id: 'DeleteSeries'
    }]
  }
  ], [])
  const onMenuItemSelect = (args) => {
    let selectedMenuItem = args.item.id;
    if (selectedTarget && selectedTarget.classList.contains('e-appointment'))
    {
      setEventObj(scheduleObj.current.getEventDetails(selectedTarget));
    }
    switch (selectedMenuItem) {

```

```

    case 'Today':
        scheduleObj.current.selectedDate = new Date();
        break;
    case 'Add':
    case 'AddRecurrence':
        let selectedCells = scheduleObj.current.getSelectedElements();
        let activeCellsData =
scheduleObj.current.getCellDetails(selectedCells.length > 0 ? selectedCells :
selectedTarget);
        if (selectedMenuItem === 'Add') {
            scheduleObj.current.openEditor(activeCellsData, 'Add');
        } else {
            scheduleObj.current.openEditor(activeCellsData, 'Add', null, 1);
        }
        break;
    case 'Save':
    case 'EditOccurrence':
    case 'EditSeries':
        if (selectedMenuItem === 'EditSeries') {
            setEventObj(new
DataManager(scheduleObj.current.eventsData).executeLocal(
                new Query().where(scheduleObj.current.eventFields.id,
                    'equal',
eventObj[scheduleObj.current.eventFields.recurrenceID][0])));
        }
        scheduleObj.current.openEditor(eventObj, selectedMenuItem);
        break;
    case 'Delete':
        scheduleObj.current.deleteEvent(eventObj);
        break;
    case 'DeleteOccurrence':
    case 'DeleteSeries':
        scheduleObj.current.deleteEvent(eventObj, selectedMenuItem);
        break;
    }
}
const onContextMenuBeforeOpen = (args) => {
    let newEventElement = document.querySelector('.e-new-event');
    if (newEventElement) {
        remove(newEventElement);
        removeClass([document.querySelector('.e-selected-cell')], 'e-selected-
cell');
    }
    scheduleObj.current.closeQuickInfoPopup();
    let targetElement = args.event.target;
    if (closest(targetElement, '.e-contextmenu')) {
        return;
    }
    selectedTarget = closest(targetElement, '.e-appointment,.e-work-cells,' +
'.e-vertical-view .e-date-header-wrap .e-all-day-cells,.e-vertical-view
.e-date-header-wrap .e-header-cells');
    if (isNullOrUndefined(selectedTarget)) {
        args.cancel = true;
        return;
    }
    if (selectedTarget.classList.contains('e-appointment')) {
        setEventObj(scheduleObj.current.getEventDetails(selectedTarget));
    }

```

```

        return;
    }
    menuObj.current.hideItems(['Save', 'Delete', 'EditRecurrenceEvent',
'DeleteRecurrenceEvent'], true);
    menuObj.current.showItems(['Add', 'AddRecurrence', 'Today'], true);
    }
    useEffect(() => {
        if (eventObj && eventObj.RecurrenceRule) {
            menuObj.current.showItems(['EditRecurrenceEvent',
'DeleteRecurrenceEvent'], true);
            menuObj.current.hideItems(['Add', 'AddRecurrence', 'Today', 'Save',
'Delete'], true);
        } else {
            menuObj.current.showItems(['Save', 'Delete'], true);
            menuObj.current.hideItems(['Add', 'AddRecurrence', 'Today',
'EditRecurrenceEvent', 'DeleteRecurrenceEvent'], true);
        }
    }, [eventObj]);
    return (
        <div className='schedule-control-section'>
            <div className='control-section'>
                <div className='control-wrapper'>
                    <ScheduleComponent height='550px' ref={scheduleObj}
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
                        <ViewsDirective>
                            <ViewDirective option='Day' />
                            <ViewDirective option='Week' />
                            <ViewDirective option='WorkWeek' />
                            <ViewDirective option='Month' />
                            <ViewDirective option='Agenda' />
                        </ViewsDirective>
                        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
                    </ScheduleComponent>
                </div>
            </div>
            <ContextMenuComponent cssClass='schedule-context-menu' ref={menuObj}
target='.e-schedule' items={menuItems} beforeOpen={onContextMenuBeforeOpen}
select={onMenuItemSelect} />
        </div>
    );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef, useState, useMemo, useEffect } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { closest, isNullOrUndefined, remove, removeClass } from
'@syncfusion/ej2-base';
import { Query, DataManager } from '@syncfusion/ej2-data';
import {
    ScheduleComponent, ViewsDirective, ViewDirective, CellClickEventArgs,
    Day, Week, WorkWeek, Month, Agenda, Inject
} from '@syncfusion/ej2-react-schedule';

```

```

import { BeforeOpenCloseMenuEventArgs, MenuEventArgs, MenuItemModel,
ContextMenuComponent } from '@syncfusion/ej2-react-navigations';
import { scheduleData } from './datasource';
interface EventObject {
  [key: string]: Object;
}
}
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const menuObj = useRef<ContextMenuComponent>(null);
  const [eventObj, setEventObj] = useState<EventObject | null>(null);
  const eventSettings = { dataSource: scheduleData };
  let selectedTarget: Element;
  const menuItems: MenuItemModel[] = useMemo(() => [{
    text: 'New Event',
    iconCss: 'e-icons new',
    id: 'Add'
  }, {
    text: 'New Recurring Event',
    iconCss: 'e-icons recurrence',
    id: 'AddRecurrence'
  }, {
    text: 'Today',
    iconCss: 'e-icons today',
    id: 'Today'
  }, {
    text: 'Edit Event',
    iconCss: 'e-icons edit',
    id: 'Save'
  }, {
    text: 'Edit Event',
    id: 'EditRecurrenceEvent',
    iconCss: 'e-icons edit',
    items: [{
      text: 'Edit Occurrence',
      id: 'EditOccurrence'
    }, {
      text: 'Edit Series',
      id: 'EditSeries'
    }]
  }, {
    text: 'Delete Event',
    iconCss: 'e-icons delete',
    id: 'Delete'
  }, {
    text: 'Delete Event',
    id: 'DeleteRecurrenceEvent',
    iconCss: 'e-icons delete',
    items: [{
      text: 'Delete Occurrence',
      id: 'DeleteOccurrence'
    }, {
      text: 'Delete Series',
      id: 'DeleteSeries'
    }]
  }
  ], [])
  const onMenuItemSelect = (args: MenuEventArgs): void => {

```

```

let selectedItem: string = args.item.id;
if (selectedTarget && selectedTarget.classList.contains('e-appointment'))
{
    setEventObj(scheduleObj.current!.getEventDetails(selectedTarget) as
EventObject);
}
switch (selectedMenuItem) {
case 'Today':
    scheduleObj.current!.selectedDate = new Date();
    break;
case 'Add':
case 'AddRecurrence':
    let selectedCells: Element[] =
scheduleObj.current!.getSelectedElements();
    let activeCellsData: CellClickEventArgs =
scheduleObj.current!.getCellDetails(selectedCells.length > 0 ?
selectedCells : selectedTarget);
    if (selectedMenuItem === 'Add') {
        scheduleObj.current!.openEditor(activeCellsData, 'Add');
    } else {
        scheduleObj.current!.openEditor(activeCellsData, 'Add', null, 1);
    }
    break;
case 'Save':
case 'EditOccurrence':
case 'EditSeries':
    if (selectedMenuItem === 'EditSeries') {
        setEventObj(new
DataManager(scheduleObj.current!.eventsData).executeLocal(
            new Query().where(scheduleObj.current!.eventFields.id,
                'equal',
eventObj[scheduleObj.current!.eventFields.recurrenceID] as string |
number))[0] as EventObject);
    }
    scheduleObj.current!.openEditor(eventObj!, selectedMenuItem);
    break;
case 'Delete':
    scheduleObj.current!.deleteEvent(eventObj!);
    break;
case 'DeleteOccurrence':
case 'DeleteSeries':
    scheduleObj.current!.deleteEvent(eventObj!, selectedMenuItem);
    break;
}
}
const onContextMenuBeforeOpen = (args: BeforeOpenCloseMenuEventArgs): void
=> {
    let newEventElement: HTMLElement = document.querySelector('.e-new-event')
as HTMLElement;
    if (newEventElement) {
        remove(newEventElement);
        removeClass([document.querySelector('.e-selected-cell')], 'e-selected-
cell');
    }
    scheduleObj.current!.closeQuickInfoPopup();
    let targetElement: HTMLElement = args.event.target as HTMLElement;
    if (closest(targetElement, '.e-contextmenu')) {

```



```

        return;
    }
    selectedTarget = closest(targetElement, '.e-appointment,.e-work-cells,' +
        '.e-vertical-view .e-date-header-wrap .e-all-day-cells,.e-vertical-view' +
        '.e-date-header-wrap .e-header-cells');
    if (isNullOrUndefined(selectedTarget)) {
        args.cancel = true;
        return;
    }
    if (selectedTarget.classList.contains('e-appointment')) {
        setEventObj(scheduleObj.current!.getEventDetails(selectedTarget) as
EventObject);
        return;
    }
    menuObj.current!.hideItems(['Save', 'Delete', 'EditRecurrenceEvent',
'DeleteRecurrenceEvent'], true);
    menuObj.current!.showItems(['Add', 'AddRecurrence', 'Today'], true);
}
useEffect(() => {
    if (eventObj && eventObj.RecurrenceRule) {
        menuObj.current!.showItems(['EditRecurrenceEvent',
'DeleteRecurrenceEvent'], true);
        menuObj.current!.hideItems(['Add', 'AddRecurrence', 'Today', 'Save',
'Delete'], true);
    } else {
        menuObj.current!.showItems(['Save', 'Delete'], true);
        menuObj.current!.hideItems(['Add', 'AddRecurrence', 'Today',
'EditRecurrenceEvent', 'DeleteRecurrenceEvent'], true);
    }
}, [eventObj]);
return (
    <div className='schedule-control-section'>
        <div className='control-section'>
            <div className='control-wrapper'>
                <ScheduleComponent height='550px' ref={scheduleObj}
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
                    <ViewsDirective>
                        <ViewDirective option='Day' />
                        <ViewDirective option='Week' />
                        <ViewDirective option='WorkWeek' />
                        <ViewDirective option='Month' />
                        <ViewDirective option='Agenda' />
                    </ViewsDirective>
                    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
                </ScheduleComponent>
            </div>
            <div>
                <ContextMenuComponent cssClass='schedule-context-menu' ref={menuObj}
target='.e-schedule' items={menuItems} beforeOpen={onContextMenuBeforeOpen}
select={onMenuItemSelect} />
            </div>
        </div>
    );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

INDEX.CSS

```

.schedule-context-menu .e-menu-item .new::before {
  content: '\e7f9';
}

```

```

}
.schedule-context-menu .e-menu-item .edit::before {
  content: '\ea9a';
}
.schedule-context-menu .e-menu-item .recurrence::before {
  content: '\e308';
  font-weight: bold;
}
.schedule-context-menu .e-menu-item .today::before {
  content: '\e322';
}
.schedule-context-menu .e-menu-item .delete::before {
  content: '\e94a';
}
.e-bigger .schedule-context-menu ul .e-menu-item .e-menu-icon {
  font-size: 14px;
}
.schedule-context-menu ul .e-menu-item .e-menu-icon {
  font-size: 12px;
}
}

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Dimensions in React Schedule component

The Scheduler dimensions refers to both height and width of the entire layout and it accepts 3 types of values.

- auto
- pixel
- percentage

Auto Height and Width

When height and width of the Scheduler are set to **auto**, it will try as hard as possible to keep an element the same width as its parent container. In other words, the parent container that holds Scheduler, its width/height will be the sum of its children. By default, Scheduler is assigned with **auto** values for both height and width properties.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData }
  return <ScheduleComponent height='auto' width='auto' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));

```

```
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return <ScheduleComponent height='auto' width='auto' selectedDate={new
    Date(2018, 1, 15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
</style>
```

```

        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
        .e-past-app {
            background-color: chocolate !important;
        }
        .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
        .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Height and Width in pixel

The Scheduler height and width will be rendered exactly as per the given pixel values. It accepts both string and number values.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    return (<ScheduleComponent height='550px' width='650px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent >);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {

```

```

ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData }
  return (<ScheduleComponent height='550px' width='650px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>

```

```

    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Height and Width in percentage

When height and width of the Scheduler are given as percentage, it will make the Scheduler as wide as the parent container.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData }
    return <ScheduleComponent height='100%' width='100%' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData }
    return <ScheduleComponent height='100%' width='100%' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>

```

```

    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <link href="index.css" rel="stylesheet" />
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
    .e-past-app {
      background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,

```



```

        .custom-class.e-schedule .e-month-view .e-appointment {
            background: green;
        }
    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

See Also

- [How to Change Scheduler Cell Dimensions](#)

Recurrence editor in React Schedule component

The Recurrence editor is integrated into Scheduler editor window by default, to process the recurrence rule generation for events. Apart from this, it can also be used as an individual component referring from the Scheduler repository to work with the recurrence related processes.

All the valid recurrence rule string mentioned in the [iCalendar](#) specifications are applicable to use with the recurrence editor.

Customizing the repeat type option in editor

By default, there are 5 types of repeat options available in recurrence editor such as,

- Never
- Daily
- Weekly
- Monthly
- Yearly

It is possible to customize the recurrence editor to display only the specific repeat options such as **Daily** and **Weekly** options alone by setting the appropriate **frequencies** option.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const scheduleObj = useRef(null);
    const eventSettings = { dataSource: scheduleData };
    const onPopupOpen = (args) => {
        if (args.type === 'Editor') {

```

```

        scheduleObj.current.eventWindow.recurrenceEditor.frequencies =
        ['none', 'daily', 'weekly'];
    }
}
return (<ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
popupOpen={onPopupOpen.bind(this)}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
</ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    PopupOpenEventArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const eventSettings: EventSettingsModel = { dataSource: scheduleData }
    const onPopupOpen = (args: PopupOpenEventArgs): void => {
        if (args.type === 'Editor') {
            scheduleObj.current.eventWindow.recurrenceEditor.frequencies = ['none',
            'daily', 'weekly'];
        }
    }
    return (<ScheduleComponent ref={scheduleObj} width='100%' height='550px'
selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings}
popupOpen={onPopupOpen.bind(this)}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-past-app {
        background-color: chocolate !important;
    }
    .custom-class.e-schedule .e-vertical-view .e-all-day-appointment-
wrapper .e-appointment,
    .custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-
appointment,
    .custom-class.e-schedule .e-month-view .e-appointment {
        background: green;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

The other properties available in recurrence editor are tabulated below,

Properties	Type	Description
firstDayOfWeek	number	Sets the first day of the week.
startDate	Date	Sets the start date

- | dateFormat | string | Sets the specific date format on recurrence editor.
- | locale | string | Sets the locale to be applied on recurrence editor.
- | cssClass | string | Allows styling with custom class names.
- | enableRtl | boolean | Allows recurrence editor to render in RTL mode.
- | minDate | Date | Sets the minimum date on recurrence editor.
- | maxDate | Date | Sets the maximum date on recurrence editor.
- | value | string | Sets the recurrence rule as its output values.
- | selectedType | number | Sets the current repeat type to be set on the recurrence editor.

Customizing the End Type Option in Editor

By default, there are 3 types of end options available in the recurrence editor such as:

- Never
- Until
- Count

It is possible to customize the recurrence editor to display only the specific end options, such as the **Until** and **Count** options alone, by setting the appropriate [endTypes](#) option.

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef, useEffect } from 'react';
import { RecurrenceEditorComponent }
  from '@syncfusion/ej2-react-schedule';
const App = () => {
  const recObject = useRef(null);
  useEffect(() => {
    recObject.current.endTypes = ['until', 'count'];
  }, []);
  return (
    <div className='content-wrapper recurrence-editor-wrap'>
      <div className='RecurrenceEditor'>
        <RecurrenceEditorComponent id='RecurrenceEditor'
          ref={recObject}></RecurrenceEditorComponent>
      </div>
    </div>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef, useEffect } from 'react';
import { RecurrenceEditorComponent }
  from '@syncfusion/ej2-react-schedule';
const App = () => {
  const recObject = useRef<RecurrenceEditorComponent>(null);
```

```

useEffect(() => {
  recObject.current.endTypes = ['until', 'count'];
}, []);
return (<div className='content-wrapper recurrence-editor-wrap'>
  <div className='RecurrenceEditor'>
    <RecurrenceEditorComponent id='RecurrenceEditor'
ref={recObject}></RecurrenceEditorComponent>
  </div>
</div>)
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/22.1.34/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/22.1.34/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/22.1.34/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/22.1.34/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/22.1.34/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/22.1.34/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/22.1.34/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/22.1.34/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>

```

```

</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Accessing the recurrence rule string

The recurrence rule is usually generated based on the options selected from the recurrence editor and also it follows the [iCalendar](#) specifications. The generated recurrence rule string is a valid one to be used with the Scheduler event's recurrence rule field.

There is a **change** event available in recurrence editor, that triggers on every time the fields of recurrence editor tends to change. Within this event argument, you can access the generated recurrence value through the **value** option as shown in the following code example.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef, useEffect } from 'react';
import { RecurrenceEditorComponent } from '@syncfusion/ej2-react-schedule';
const App = () => {
  const ruleOutput = useRef(null);
  useEffect(() => {
    let outputElement = ruleOutput.current;
    outputElement.innerText = 'Select Rule';
  }, []);
  const onChange = (args) => {
    let outputElement = ruleOutput.current;
    if (args.value == "") {
      outputElement.innerText = 'Select Rule';
    }
    else {
      outputElement.innerText = args.value;
    }
  }
  return (
    <div className='content-wrapper recurrence-editor-wrap'>
      <div style={{ paddingBottom: '15px' }}>
        <label>Rule Output</label>
        <div className='rule-output-container'>
          <div ref={ruleOutput}></div>
        </div>
      </div>
      <div className='RecurrenceEditor'>
        <RecurrenceEditorComponent id='RecurrenceEditor'
          change={onChange}></RecurrenceEditorComponent>
      </div>
    </div>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef, useEffect } from 'react';
import { RecurrenceEditorComponent, RecurrenceEditorChangeEventArgs }
  from '@syncfusion/ej2-react-schedule';
const App = () => {
  const ruleOutput = useRef(null);
  useEffect(() => {
    let outputElement: HTMLElement = ruleOutput.current
    outputElement.innerText = 'Select Rule';
  }, []);
  const onChange = (args: RecurrenceEditorChangeEventArgs): void => {
    let outputElement: HTMLElement = ruleOutput.current;
    if (args.value === "") {
      outputElement.innerText = 'Select Rule';
    } else {
      outputElement.innerText = args.value;
    }
  }
  return (<div className='content-wrapper recurrence-editor-wrap'>
    <div style={{ paddingBottom: '15px' }}>
      <label>Rule Output</label>
      <div className='rule-output-container'>
        <div ref={ruleOutput}></div>
      </div>
    </div>
    <div className='RecurrenceEditor'>
      <RecurrenceEditorComponent id='RecurrenceEditor'
change={onChange}></RecurrenceEditorComponent>
    </div>
  </div>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Set specific value on recurrence editor

It is possible to display the recurrence editor with specific options loaded initially, based on the rule string that we provide. The fields of recurrence editor will change its values accordingly, when we provide a particular rule through the `setRecurrenceRule` method.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef, useEffect } from 'react';
import { RecurrenceEditorComponent, RecurrenceEditorChangeEventArgs }
    from '@syncfusion/ej2-react-schedule';
const App = () => {
    const recObject = useRef(null);
    const ruleOutput = useRef(null);
    useEffect(() => {
        let outputElement = ruleOutput.current;
        recObject.current.setRecurrenceRule('FREQ=DAILY; INTERVAL=2; COUNT=8');
        outputElement.innerText = recObject.current.value;
    }, []);
    const onChange = (args) => {
        let outputElement = ruleOutput.current;

```



```

    outputElement.innerText = args.value;
  }
  return (<div className='content-wrapper recurrence-editor-wrap'>
    <div style={{ paddingBottom: '15px' }}>
      <label>Rule Output</label>
      <div className='rule-output-container'>
        <div ref={ruleOutput}></div>
      </div>
    </div>
    <div className='RecurrenceEditor'>
      <RecurrenceEditorComponent id='RecurrenceEditor' ref={recObject}
change={onChange}></RecurrenceEditorComponent>
    </div>
  </div>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef, useEffect } from 'react';
import { RecurrenceEditorComponent, RecurrenceEditorChangeEventArgs }
  from '@syncfusion/ej2-react-schedule';
const App = () => {
  const recObject = useRef<RecurrenceEditorComponent>(null);
  const ruleOutput = useRef(null);
  useEffect(() => {
    let outputElement: HTMLElement = ruleOutput.current;
    recObject.current.setRecurrenceRule('FREQ=DAILY; INTERVAL=2; COUNT=8');
    outputElement.innerText = recObject.current.value as string;
  }, []);
  const onChange = (args: RecurrenceEditorChangeEventArgs): void => {
    let outputElement: HTMLElement = ruleOutput.current;
    outputElement.innerText = args.value;
  }
  return (<div className='content-wrapper recurrence-editor-wrap'>
    <div style={{ paddingBottom: '15px' }}>
      <label>Rule Output</label>
      <div className='rule-output-container'>
        <div ref={ruleOutput}></div>
      </div>
    </div>
    <div className='RecurrenceEditor'>
      <RecurrenceEditorComponent id='RecurrenceEditor' ref={recObject}
change={onChange}></RecurrenceEditorComponent>
    </div>
  </div>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```
<!DOCTYPE html>
```

```

<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Recurrence date generation

You can parse the `recurrenceRule` of an event to generate the date instances on which that particular event is going to occur, using the `getRecurrenceDates` method. It generates the dates based on the `recurrenceRule` that we provide. The parameters to be provided for `getRecurrenceDates` method are as follows.

Field name	Type	Description
----- ----- -----		
startDate	Date	Appointment start date.
rule	String	Recurrence rule present in an event object.
excludeDate	String	Date collection (in ISO format) to be excluded. It is optional .
maximumCount	Number	Number of date count to be generated. It is optional .
viewDate	Date	Current view range's first date. It is optional .

INDEX.JSX

```
import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef, useEffect } from 'react';
import { RecurrenceEditorComponent }
  from '@syncfusion/ej2-react-schedule';
const App = () => {
  const recObject = useRef(null);
  const ruleOutput = useRef(null);
  const today = new Date()
  useEffect(() => {
    let dates = recObject.current.getRecurrenceDates(new
Date(today.getFullYear(), today.getMonth(), today.getDate(), 10, 0),
    'FREQ=DAILY;INTERVAL=1', '20180108T114224Z,20180110T114224Z', 4, new
Date(today.getFullYear(), today.getMonth(), today.getDate()));
    let stringCollection = '';
    for (let index = 0; index < dates.length; index++) {
      stringCollection += new Date(dates[index]);
      if (index + 1 !== dates.length) {
        stringCollection += '\n';
      }
    }
    let outputElement = ruleOutput.current;
    outputElement.innerText = stringCollection;
  }, []);
  return (<div className='content-wrapper recurrence-editor-wrap'>
    <div style={{ paddingBottom: '15px' }}>
      <label>Date Collections</label>
      <div className='rule-output-container'>
        <div ref={ruleOutput}></div>
      </div>
      <div className='RecurrenceEditor' style={{ display: "none" }}>
        <RecurrenceEditorComponent id='RecurrenceEditor'
ref={recObject}></RecurrenceEditorComponent>
      </div>
    </div>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef, useEffect } from 'react';
import { RecurrenceEditorComponent }
  from '@syncfusion/ej2-react-schedule';
const App = () => {
  const recObject = useRef<RecurrenceEditorComponent>(null);
  const ruleOutput = useRef(null);
  const today: Date = new Date()
  useEffect(() => {
    let dates: number[] = recObject.current.getRecurrenceDates(new
Date(today.getFullYear(), today.getMonth(), today.getDate(), 10, 0),
    'FREQ=DAILY;INTERVAL=1', '20180108T114224Z,20180110T114224Z', 4, new
Date(today.getFullYear(), today.getMonth(), today.getDate()));
    let stringCollection: string = '';
    for (let index: number = 0; index < dates.length; index++) {
      stringCollection += new Date(dates[index]);
      if (index + 1 !== dates.length) {
        stringCollection += '\n';
      }
    }
    let outputElement: HTMLElement = ruleOutput.current;
    outputElement.innerHTML = stringCollection;
  }, []);
  return (<div className='content-wrapper recurrence-editor-wrap'>
    <div style={{ paddingBottom: '15px' }}>
      <label>Date Collections</label>
      <div className='rule-output-container'>
        <div ref={ruleOutput}></div>
      </div>
      <div className='RecurrenceEditor' style={{ display: "none" }}>
        <RecurrenceEditorComponent id='RecurrenceEditor'
ref={recObject}></RecurrenceEditorComponent>
      </div>
    </div>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<link href="index.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Above example will generate two dates January 7, 2018 & January 9 2018 by excluding the in between dates January 8 2018 & January 10 2018, since those dates were given in the exclusion list. Generated dates can then be utilised to create appointments.

Recurrence date generation in server-side

It is also possible to generate recurrence date instances from server-side by manually referring the `RecurrenceHelper` class, which is specifically written and referred from application end to handle this date generation process.

Refer [here](#) for the step by step procedure to achieve date generation in server-side.

Restrict date generation with specific count

In case, if the rule is given in "NEVER ENDS" category, then you can mention the maximum count when you actually want to stop the date generation starting from the provided start date. To do so, provide the appropriate `maximumCount` value within the `getRecurrenceDates` method as shown in the following code example.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef, useEffect } from 'react';
import { RecurrenceEditorComponent } from '@syncfusion/ej2-react-schedule';
const App = () => {
  const recObject = useRef(null);
  const ruleOutput = useRef(null);
  useEffect(() => {
    let dates = recObject.current.getRecurrenceDates(new Date(2018, 0, 7, 10, 0), 'FREQ=DAILY;INTERVAL=1', null, 10, null);
    let stringCollection = '';
    for (let index = 0; index < dates.length; index++) {
      stringCollection += new Date(dates[index]);
      if (index + 1 !== dates.length) {
        stringCollection += '\n';
      }
    }
    let outputElement = ruleOutput.current;
    outputElement.innerText = stringCollection;
  }, []);
  return (
    <div className='content-wrapper recurrence-editor-wrap'>
      <div style={{ paddingBottom: '15px' }}>
        <label>Date Collections</label>
        <div className='rule-output-container'>
          <div ref={ruleOutput}></div>
        </div>
      </div>
      <div className='RecurrenceEditor' style={{ display: 'none' }}>
        <RecurrenceEditorComponent id='RecurrenceEditor'
        ref={recObject}></RecurrenceEditorComponent>
      </div>
    </div>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef, useEffect } from 'react';
import { RecurrenceEditorComponent }
  from '@syncfusion/ej2-react-schedule';
const App = () => {
  const recObject = useRef<RecurrenceEditorComponent>(null);
  const ruleOutput = useRef(null);
  useEffect(() => {
    let dates: number[] = recObject.current.getRecurrenceDates(new Date(2018, 0, 7, 10, 0), 'FREQ=DAILY;INTERVAL=1', null, 10, null);
    let stringCollection: string = '';
    for (let index: number = 0; index < dates.length; index++) {
      stringCollection += new Date(dates[index]);
      if (index + 1 !== dates.length) {
        stringCollection += '\n';
      }
    }
  }
}

```

```

    let outputElement: HTMLElement = ruleOutput.current;
    outputElement.innerText = stringCollection;
  }, []);
  return (<div className='content-wrapper recurrence-editor-wrap'>
    <div style={{ paddingBottom: '15px' }}>
      <label>Date Collections</label>
      <div className='rule-output-container'>
        <div ref={ruleOutput}></div>
      </div>
    </div>
    <div className='RecurrenceEditor' style={{ display: 'none' }}>
      <RecurrenceEditorComponent id='RecurrenceEditor'
ref={recObject}></RecurrenceEditorComponent>
    </div>
  </div>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <link href="index.css" rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;

```

```

        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Localization in React Schedule component

The Scheduler integrates different date-time formats and cultures, which allows it to function globally, thus meeting the diverse needs of different regions.

You can adapt the Scheduler to various languages by parsing and formatting the date or number ([Internationalization](#)), adding culture specific customization and translation to the text ([Localization](#)).

Globalization

The Internationalization library provides support for formatting and parsing the number, date, and time by using the official [Unicode CLDR](#) JSON data and also provides the `loadCldr` method to load the culture specific CLDR JSON data.

By default, Scheduler is set to follow the English culture ('en-US'). If you want to go with different culture other than English, follow the below steps.

- Install the `CLDR-Data` package by using the below command (it installs the CLDR JSON data). For more information about CLDR-Data, refer to this [link](#).

```
npm install cldr-data --save
```

Once the package is installed, you can find the culture specific JSON data under the location `/node_modules/cldr-data`.

- Now import the installed CLDR JSON data into the `app.ts` file. To import JSON data, you need to install the JSON plugin loader. Here, we have used the SystemJS JSON plugin loader.

```
npm install systemjs-plugin-json --save-dev
```


- Once installed, configure the `system.config.js` configuration settings as shown in the following code to map the `systemjs-plugin-json` loader.

```
`ts
System.config({
  paths: {
    'syncfusion': 'npm:@syncfusion/'
  },
  map: {
    app: 'app',
    //Syncfusion packages mapping
    "@syncfusion/ej2-base": "syncfusion:ej2-base/dist/ej2-base.umd.min.js",
    "@syncfusion/ej2-data": "syncfusion:ej2-data/dist/ej2-data.umd.min.js",
    "@syncfusion/ej2-schedule": "syncfusion:ej2-schedule/dist/ej2-schedule.umd.min.js",
    "@syncfusion/ej2-calendars": "syncfusion:ej2-calendars/dist/ej2-calendars.umd.min.js",
    "@syncfusion/ej2-buttons": "syncfusion:ej2-buttons/dist/ej2-buttons.umd.min.js",
    "@syncfusion/ej2-inputs": "syncfusion:ej2-inputs/dist/ej2-inputs.umd.min.js",
    "@syncfusion/ej2-lists": "syncfusion:ej2-lists/dist/ej2-lists.umd.min.js",
    "@syncfusion/ej2-navigations": "syncfusion:ej2-navigations/dist/ej2-navigations.umd.min.js",
    "@syncfusion/ej2-popups": "syncfusion:ej2-popups/dist/ej2-popups.umd.min.js",
    "@syncfusion/ej2-dropdowns": "syncfusion:ej2-dropdowns/dist/ej2-dropdowns.umd.min.js",
    "@syncfusion/ej2-splitbuttons": "syncfusion:ej2-splitbuttons/dist/ej2-splitbuttons.umd.min.js",
    "cldr-data": 'npm:cldr-data',
    "plugin-json": "npm:systemjs-plugin-json/json.js"
  },
  meta: {
    '*.json': { loader: 'plugin-json' }
  },
  packages: {
    'app': { main: 'app', defaultExtension: 'js' },
    'cldr-data': { main: 'index.js', defaultExtension: 'js' }
  }
});
System.import('app');
```

- Now import the required cultures from the installed location to `app.ts` file as given in the following code example.

```
`ts
//import the loadCldr from ej2-base
import { loadCldr } from '@syncfusion/ej2-base';
loadCldr(
  require('cldr-data/supplemental/numberingSystems.json'),
  require('cldr-data/main/fr-CH/ca-gregorian.json'),
  require('cldr-data/main/fr-CH/numbers.json'),
  require('cldr-data/main/fr-CH/timeZoneNames.json')
);`
```

- Set the culture to Scheduler by using the `locale` property.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
import { loadCldr } from '@syncfusion/ej2-base';
import * as numberingSystems from '../numberingSystems.json';
import * as gregorian from '../ca-gregorian.json';
import * as numbers from '../numbers.json';
import * as timeZoneNames from '../timeZoneNames.json';
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)} locale='fr-CH' eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
import { loadCldr } from '@syncfusion/ej2-base';
import * as numberingSystems from '../numberingSystems.json';
import * as gregorian from '../ca-gregorian.json';
import * as numbers from '../numbers.json';
import * as timeZoneNames from '../timeZoneNames.json';
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (
    <ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)} locale='fr-CH' eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-popups/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Localizing the static Scheduler text

[Localization](#) library allows to display all the static text, date content, and time mode of the Scheduler following the localized language. To achieve this, set the `locale` property of Scheduler, as well as define the translation text of static words of Scheduler through the `load` method.

For example, the following code example lets you to define the French translation words for all the static words used in Scheduler.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
import { Ajax, L10n, loadCldr } from '@syncfusion/ej2-base';
import * as numberingSystems from '../numberingSystems.json';
import * as gregorian from '../ca-gregorian.json';
import * as numbers from '../numbers.json';
import * as detimeZoneNames from '../timeZoneNames.json';
loadCldr(numberingSystems, gregorian, numbers, detimeZoneNames);
let localeTexts;
let ajax = new Ajax('../locale.json', 'GET', false);
ajax.onSuccess = (value) => {
  localeTexts = value;
};
ajax.send();
L10n.load(JSON.parse(localeTexts));
const App = () => {

```

```

    const eventSettings = { dataSource: scheduleData }
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} locale='fr-CH' eventSettings={eventSettings}>
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>;
  }
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
import { Ajax, L10n, loadCldr } from '@syncfusion/ej2-base';
import * as numberingSystems from '../numberingSystems.json';
import * as gregorian from '../ca-gregorian.json';
import * as numbers from '../numbers.json';
import * as detimeZoneNames from '../timeZoneNames.json';
loadCldr(numberingSystems, gregorian, numbers, detimeZoneNames);
let localeTexts: string;
let ajax: Ajax = new Ajax('../locale.json', 'GET', false);
ajax.onSuccess = (value: string) => {
  localeTexts = value;
};
ajax.send();
L10n.load(JSON.parse(localeTexts));
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData};
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} locale='fr-CH' eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

The localized words for static text used in Scheduler and Recurrence Editor can be referred from the following code.

```
`ts
```

```
L10n.load({
```

```
"en": {  
  "schedule": {  
    "day": "Day",  
    "week": "Week",  
    "workWeek": "Work Week",  
    "month": "Month",  
    "agenda": "Agenda",  
    "weekAgenda": "Week Agenda",  
    "workWeekAgenda": "Work Week Agenda",  
    "monthAgenda": "Month Agenda",  
    "today": "Today",  
    "noEvents": "No events",  
    "emptyContainer": "There are no events scheduled on this day.",  
    "allDay": "All day",  
    "start": "Start",  
    "end": "End",  
    "more": "more",  
    "close": "Close",  
    "cancel": "Cancel",  
    "noTitle": "(No Title)",  
    "delete": "Delete",  
    "deleteEvent": "Delete Event",  
    "deleteMultipleEvent": "Delete Multiple Events",  
    "selectedItems": "Items selected",  
    "deleteSeries": "Delete Series",  
    "edit": "Edit",  
    "editSeries": "Edit Series",  
    "editEvent": "Edit Event",  
    "createEvent": "Create",  
    "subject": "Subject",  
    "addTitle": "Add title",  
    "moreDetails": "More Details",  
    "save": "Save",
```

"editContent": "Do you want to edit only this event or entire series?",
"deleteRecurrenceContent": "Do you want to delete only this event or entire series?",
"deleteContent": "Are you sure you want to delete this event?",
"deleteMultipleContent": "Are you sure you want to delete the selected events?",
"newEvent": "New Event",
"title": "Title",
"location": "Location",
"description": "Description",
"timezone": "Timezone",
"startTimezone": "Start Timezone",
"endTimezone": "End Timezone",
"repeat": "Repeat",
"saveButton": "Save",
"cancelButton": "Cancel",
"deleteButton": "Delete",
"recurrence": "Recurrence",
"wrongPattern": "The recurrence pattern is not valid.",
"seriesChangeAlert": "The changes made to specific instances of this series will be cancelled and those events will match the series again.",
"createError": "The duration of the event must be shorter than how frequently it occurs. Shorten the duration, or change the recurrence pattern in the recurrence event editor.",
"recurrenceDateValidation": "Some months have fewer than the selected date. For these months, the occurrence will fall on the last date of the month.",
"sameDayAlert": "Two occurrences of the same event cannot occur on the same day.",
"editRecurrence": "Edit Recurrence",
"repeats": "Repeats",
"alert": "Alert",
"startEndError": "The selected end date occurs before the start date.",
"invalidDateError": "The entered date value is invalid.",
"ok": "Ok",
"occurrence": "Occurrence",
"series": "Series",
"previous": "Previous",
"next": "Next",


```
"timelineDay": "Timeline Day",
"timelineWeek": "Timeline Week",
"timelineWorkWeek": "Timeline Work Week",
"timelineMonth": "Timeline Month",
"expandAllDaySection": "Expand",
"collapseAllDaySection": "Collapse"
},
"recurrenceeditor": {
  "none": "None",
  "daily": "Daily",
  "weekly": "Weekly",
  "monthly": "Monthly",
  "month": "Month",
  "yearly": "Yearly",
  "never": "Never",
  "until": "Until",
  "count": "Count",
  "first": "First",
  "second": "Second",
  "third": "Third",
  "fourth": "Fourth",
  "last": "Last",
  "repeat": "Repeat",
  "repeatEvery": "Repeat Every",
  "on": "Repeat On",
  "end": "End",
  "onDay": "Day",
  "days": "Day(s)",
  "weeks": "Week(s)",
  "months": "Month(s)",
  "years": "Year(s)",
  "every": "every",
  "summaryTimes": "time(s)",
```

```

"summaryOn": "on",
"summaryUntil": "until",
"summaryRepeat": "Repeats",
"summaryDay": "day(s)",
"summaryWeek": "week(s)",
"summaryMonth": "month(s)",
"summaryYear": "year(s)"
}
}
});
`

```

Setting date format

Scheduler can be used with all valid date formats and by default it follows the universal date format "MM/dd/yyyy". If the `dateFormat` property is not specified particularly, then it will work based on the locale that is assigned to the Scheduler. As the default locale applied on Scheduler is "en-US", this makes it to follow the "MM/dd/yyyy" pattern.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: scheduleData };
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} dateFormat="yyyy/MM/dd" eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
}

```

```

} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} dateFormat="yyyy/MM/dd" eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdn.jsdelivr.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;

```

```

        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Setting the time format

Time formats is a way of representing the time value in different string formats in the Scheduler. By default, the time mode of the Scheduler can be either 12 or 24 hours format which is completely based on the **locale** set to the Scheduler. Since the default **locale** value of the Scheduler is en-US, the time mode will be set to 12 hours format automatically. You can also customize the format by using the **timeFormat** property. To know more about the time format standards, refer to the [Date and Time Format](#) section.

The following example demonstrates the Scheduler component in 24 hours format.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} timeFormat="HH:mm" eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
    ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';

```

```
import { scheduleData } from './datasource';
const App = () => {
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1, 15)} timeFormat="HH:mm" eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdn.jsdelivr.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></script>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
```

```

        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Note: `timeFormat` property only accepts the valid time format's.

First day of the week

By default, the first day of the week can be set on Scheduler by making use of the `firstDayOfWeek` property, doing so will make the Scheduler to start with that day.

Here, Sunday is always denoted as 0, Monday as 1 and so on.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Month, Week, WorkWeek, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} firstDayOfWeek={3} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Week, WorkWeek, Month]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Month, Week, WorkWeek, Inject,
    ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    return (<ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} firstDayOfWeek={3} eventSettings={eventSettings}>

```

```

    <ViewsDirective>
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Week, WorkWeek, Month]} />
  </ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>

```

```

        <div id='schedule'>
            <div id='loader'>Loading....</div>
        </div>
    </body>
</html>

```

Displaying Scheduler in RTL mode

The Scheduler layout and its behavior can be changed as per the common RTL (Right to Left) conventions by setting `enableRtl` to `true`. By doing so, the Scheduler will display its usual layout from right to left. Its default value is `false`.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Inject, ViewsDirective,
ViewDirective } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} enableRtl={true} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek]} />
    </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Inject,
    ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    return <ScheduleComponent height='550px' selectedDate={new Date(2018, 1,
15)} enableRtl={true} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek]} />
    </ScheduleComponent>
};

```



```
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='schedule'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

See Also

- [How to change first day of the week in the Scheduler](#)

Accessibility in React Schedule component

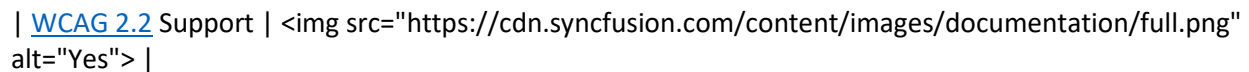
The Scheduler has been designed based on the WAI-ARIA specifications, thus applying the appropriate ARIA roles, states and properties for the Scheduler elements. It is also available with a built-in keyboard navigation support, making it easier for the people who use assistive technologies or who completely rely on the Keyboard support. As per the accessibility standard, the navigated dates, views and other interactive actions performed on the Scheduler will be read out to the target users who use assistive technologies such as screen readers.

The Scheduler makes use of the most required ARIA attributes such as `aria-label` and `role` to support the accessibility in it. To be more accurate, it must be used with an ARIA compliant browser along with the screen reader running from backend.

The accessibility compliance for the Schedule component is outlined below.

| Accessibility Criteria | Compatibility |

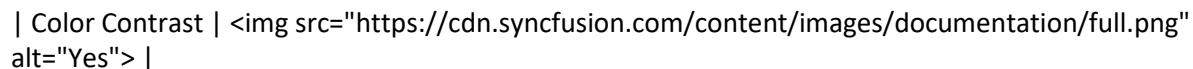
| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

```
.post .post-content img {
display: inline-block;
```

```
margin: 0.5em 0;
}
```

```
</style>
```

```
<div> - All
features of the component meet the requirement.</div>
```

```
<div> - Some features of the component do not meet the requirement.</div>
```

```
<div> - The component does not meet the requirement.</div>
```

ARIA attributes

The Scheduler parent element is assigned with a role of **main**, to denote it as the main content of a component as well as a unique element of the entire document.

The following ARIA attributes are used in the Scheduler.

Attributes	Description
------------	-------------

-----	-----
-------	-------

role="main"	Attribute added to the Scheduler element describes the actual role of the element and denote it as a main and unique content.
-------------	---

role="button"	Attribute is assigned to the appointments of Scheduler, to denote it as a clickable element.
---------------	--

aria-label	Attribute is set to the Scheduler parent element and its default value is Scheduler's current date. On every time, the date is navigated, this attribute is updated with appropriate current date values. It is also assigned to other scheduler UI elements such as previous and next date navigation buttons depicting its purpose, div element displaying date range in the header bar and appointment elements.
------------	---

aria-labelledby	It indicates editor dialog title to the user through assistive technologies.
-----------------	--

aria-describedby	It indicates editor dialog content description to the user through assistive technologies.
------------------	--

aria-disabled	Attribute is set to the appointment element to indicates the disabled state of the Scheduler.
---------------	---

Keyboard interaction

All the Scheduler actions can be controlled via keyboard keys by using the **allowKeyboardInteraction** property which is set to **true** by default. The following are the standard keys that work on Scheduler.

Keys	Description
------	-------------

-----	-----
-------	-------

Alt + j	Focuses the Scheduler element [provided from application end].
---------	--

Tab	Focuses the first or active item on the Scheduler header bar and then move the focus to the next available event elements. If no events present, then focus moves out of the component.
-----	---

| Shift + Tab | Reverse focusing of the Tab key functionality. Inverse focusing of event elements from the last one and then move onto the first or active item on Scheduler header bar and then moves out of the component.

| Enter | Opens the quick info popup on the selected cells or events. |

| Escape | Closes any of the popup that are in open state. |

| Arrow | To move onto the next available cells in either of the needed directions. (left, right, top and right) |

| Shift + Arrow | For multiple cell selection on either direction. |

| Delete | Deletes one or more selected events. |

| Ctrl + Click on events | To select multiple events. |

| Alt + Number (from 1 to 6) | To switch between the views of Scheduler. |

| Ctrl + Left Arrow | To navigate to the previous date period. |

| Ctrl + Right Arrow | To navigate to the next date period. |

| Left or Right Arrow | On pressing any of these keys, when focus is currently on the Schedule header bar, moves the focus to the previous or next items in the header bar. |

| Space or Enter | It activates any of the focused items. |

| Page Up & Page Down | To scroll through the work cells area. |

| Home | To move the selection to the first cell of Scheduler. |

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Ensuring accessibility

The Scheduler component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Scheduler component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Scheduler component with accessibility tools.

See also

- [Accessibility in Syncfusion React components](#)

Scheduler styling in React Schedule component

To modify the Scheduler appearance, you need to override the default CSS of Scheduler. Also, you have an option to create your own custom theme using our [Theme Studio](#). Please find the list of CSS classes in Scheduler.

| Css class | Purpose |

|-----|-----|

| .e-schedule .e-vertical-view .e-work-cells | Work cells in vertical views of scheduler |

| .e-schedule .e-month-view .e-work-cells | Work cells in month view of scheduler |

| .e-schedule .e-month-view .e-other-month | Work cells of other month in month view of scheduler |

| .e-schedule .e-timeline-view .e-work-cells | Work cells in timeline views of scheduler |

| .e-schedule .e-timeline-month-view .e-work-cells | Work cells in timeline month view of scheduler |

| .e-schedule .e-timeline-year-view .e-work-cells | Work cells in timeline year view of scheduler |

| .e-schedule .e-timeline-year-view .e-work-cells.e-other-month | Work cells of other month in timeline year view of scheduler |

| .e-schedule .e-month-agenda-view .e-work-cells | Work cells in month agenda view of scheduler |

| .e-schedule .e-month-agenda-view .e-other-month | Work cells of other month in month agenda view of scheduler |

| .e-schedule .e-year-view .e-calendar-wrapper .e-month-calendar.e-calendar .e-other-month | Work cells of other month in year view of scheduler |

| .e-schedule .e-vertical-view .e-all-day-cells | All day cells in vertical views of scheduler |

| .e-schedule .e-vertical-view .e-work-hours | Work hour cells in vertical views of scheduler |

| .e-schedule .e-month-view .e-work-days | Work day cells in month view of scheduler |

| .e-schedule .e-month-agenda-view .e-work-days | Work day cells in month agenda view of scheduler |

| .e-schedule .e-timeline-view .e-work-hours | Work hour cells in timeline views of scheduler |

| .e-schedule .e-timeline-month-view .e-work-days | Work day cells in timeline month view of scheduler |

| .e-schedule .e-timeline-year-view .e-work-cells.e-work-days | Work day cells in timeline year view of scheduler |

| .e-schedule .e-vertical-view .e-day-wrapper .e-appointment | Appointment in vertical views of scheduler |

| .e-schedule .e-vertical-view .e-all-day-appointment-wrapper .e-appointment | All day Appointment in vertical views of scheduler |

| .e-schedule .e-month-view .e-appointment | Appointment in month view of scheduler |

| .e-schedule .e-timeline-view .e-appointment | Appointment in timeline views of scheduler |

| .e-schedule .e-timeline-month-view .e-appointment | Appointment in timeline month view of scheduler |

| .e-schedule .e-timeline-year-view .e-event-table .e-appointment | Appointment in timeline year view of scheduler |

| .e-schedule .e-year-view .e-calendar-wrapper .e-month-calendar.e-calendar .e-appointment | Appointment in year view of scheduler |

| .e-schedule .e-agenda-view .e-appointment | Appointment in agenda view of scheduler |

| .e-schedule .e-month-agenda-view .e-appointment-indicator | Appointment in month agenda view of scheduler |

| .e-schedule .e-block-appointment | Block appointment in scheduler |

- | .e-schedule .e-read-only | Read only appointment in scheduler. |
- | e-appointment-border | Appointment which are currently selected, use the appointment class hierarchical based on your views. |
- | e-selected-cells | work cells which are currently selected, use the work cell class hierarchical based on your views. |
- | e-header-cells | Header cells of scheduler, use the work cells hierarchical based on your views. |
- | .e-schedule .e-vertical-view .e-resource-cells| Resource cells in vertical views of scheduler. |
- | .e-schedule .e-month-view .e-resource-cells| Resource cells in month view of scheduler. |
- | .e-schedule .e-timeline-view .e-resource-cells | Resource cells in timeline views of scheduler. |
- | .e-schedule .e-timeline-month-view .e-resource-cells| Resource cells in timeline month view of scheduler. |
- | e-parent-node | Parent resource cells in timeline views of scheduler. |
- | e-child-node | Child resource cells in timeline views of scheduler. |

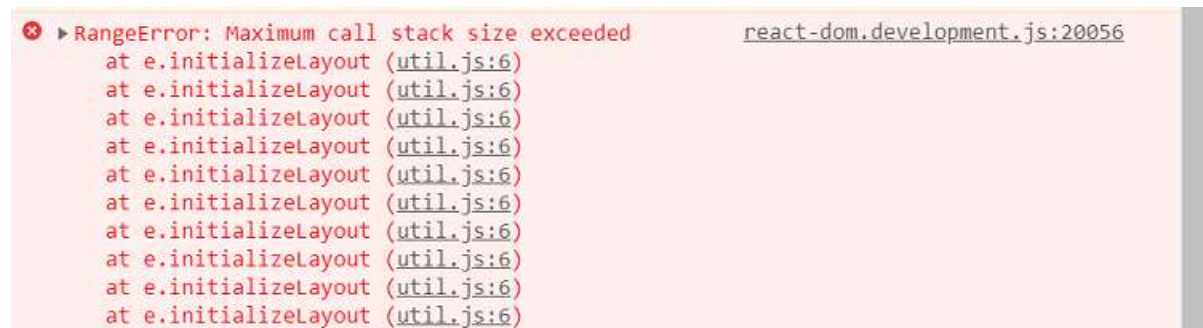
You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

Frequently asked questions in React Schedule component

In this article, you can find some frequently asked questions and corresponding solutions while getting hands-on experience with scheduler control.

Maximum call stack size exceeded

Error Image:



Solution:

The above error occurs when using scheduler views that were not imported into the project. You can resolve this issue by importing the required view modules.

In the below code, `Day` option is used without injecting, So, it throws the above error. You can resolve this problem by simply injecting the day module in below code.

```
`ts
import { render } from 'react-dom';
import * as React from 'react';
```

```

import { ScheduleComponent, ViewsDirective, ViewDirective, Agenda, TimelineViews, TimelineMonth,
Inject, Resize, DragAndDrop
} from '@syncfusion/ej2-react-schedule';
import { extend } from '@syncfusion/ej2-base';
import { SampleBase } from './sample-base';
import * as dataSource from './datasource.json';
export class TimelineView extends SampleBase {
  constructor() {
    super(...arguments);
    this.data = extend([], dataSource.scheduleData.concat(dataSource.timelineData), null, true);
  }
  private eventSettings: EventSettingsModel = { dataSource: this.data };
  render() {
    return (<ScheduleComponent height="650px" selectedDate={new Date(2021, 0, 10)}
    eventSettings={this.eventSettings}>
    <ViewsDirective>
    <ViewDirective option="Day" />
    <ViewDirective option="TimelineWeek" />
    <ViewDirective option="TimelineWorkWeek" />
    <ViewDirective option="TimelineDay" />
    <ViewDirective option="Agenda" />
    </ViewsDirective>
    <Inject
    services={[ TimelineViews, TimelineMonth, Agenda, Resize, DragAndDrop]}
    />
    </ScheduleComponent>
    );
  }
}
render(<TimelineView />, document.getElementById('sample'));

```

Grouping with empty resources

Grouping without providing any resource data will throw the following problems.

- Normal(vertical) views are rendered, but you are not able to perform CRUD operations

- Timeline views not at all render and shows empty scheduler table

So, we suggest to avoid grouping with empty resources in the scheduler.

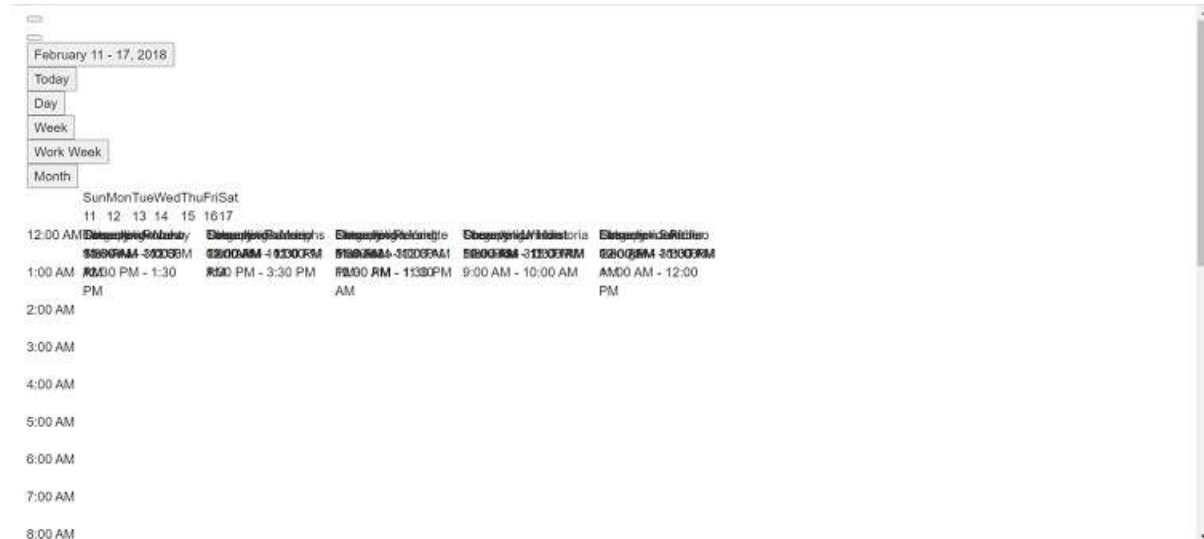
Not providing e-field in editor template

Error: While using editor template, value of `e-field` is missing in editor window.

Solution: `e-field` value is mandatory, we need to add it. Please refer [here](#) for more info.

Missing CSS reference

Error Image:



Solution:

The above problem occurs when missing CSS references for the scheduler in a project. You can resolve this issue by providing proper CSS for the scheduler.

```
<html>
<head>
<title>Syncfusion React Sample</title>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no" />
<meta http-equiv="x-ua-compatible" content="ie=edge">
<meta name="description" content="Syncfusion React UI Components" />
<meta name="author" content="Syncfusion" />
<!-- scheduler CSS is referred from this link -->
<link href="https://cdn.syncfusion.com/ej2/material.css" rel="stylesheet">
</head>
<body class="material">
```



```
<div id='sample'>
</body>
</html>
`
```

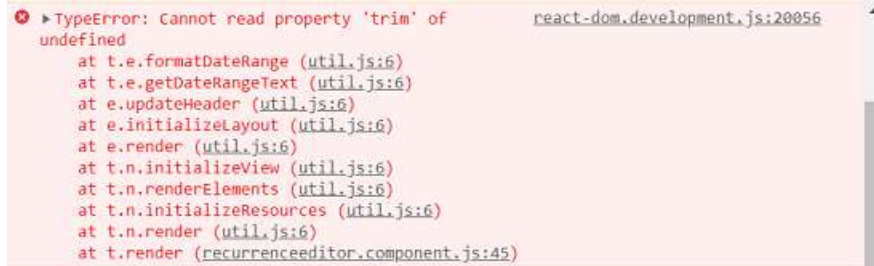
QuickInfoTemplate at bottom

When using the `quickInfoTemplate` in scheduler, sometimes quickinfo popup not shown fully at the bottom area of scheduler. You can resolve this by using `cellClick` and `eventClick` events and below code snippet.

```
`ts
constructor() {
super(...arguments);
this.eventAdded = false;
}
.
.
onClick(args) {
if (!this.eventAdded) {
let popupInstance = document.querySelector('.e-quick-popup-wrapper').ej2_instances[0];
popupInstance.open = () => {
popupInstance.refreshPosition();
};
this.eventAdded = true;
}
}
.
.
.
<ScheduleComponent id="schedule" cellClick={this.onClick.bind(this)}
eventClick={this.onClick.bind(this)}>
`
```

Not importing culture files while using localization

Error Image:



While using [locale](#) in scheduler, not importing the required culture files properly throws the problem.

Solution: Properly add and import the culture files (numberingSystems, timeZoneNames, loadCldr, L10n etc.,) in your project will resolve the problem.

`ts

```
import { loadCldr, L10n } from '@syncfusion/ej2-base';
import * as numberingSystems from './culture-files/numberingSystems.json';
import * as gregorian from './culture-files/ca-gregorian.json';
import * as numbers from './culture-files/numbers.json';
import * as timeZoneNames from './culture-files/timeZoneNames.json';
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
L10n.load({
  'en-GB': {
    schedule: {
      day: 'Day',
      week: 'Week',
      workWeek: 'Work Week',
      month: 'Month'
    }
  }
});
```

Ej1 api migration in React Schedule component

This topic shows the API equivalent of JS2 Scheduler component to be used, while migrating your project that uses JS1 Scheduler.

Scheduler

Properties

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| To change the display of days count in agenda view | **Property:** *daysInAgenda*

 <EJ.Schedule id="Schedule" currentView={ej.Schedule.CurrentView.Agenda}
 agendaViewSettings-daysInAgenda={5}>
</EJ.Schedule> | **Property:** *agendaDaysCount*

 <ScheduleComponent currentView='Agenda'
 agendaDaysCount={7}>
</ScheduleComponent> |

| Preventing deletion of appointment | **Property:** *allowDelete*

 <EJ.Schedule id="Schedule"
 allowDelete= {false}>
</EJ.Schedule> | Not applicable |

| Allows dragging and dropping of appointments | **Property:** *allowDragAndDrop*

 <EJ.Schedule id="Schedule" allowDragAndDrop={true}>
 </EJ.Schedule> | **Property:**
allowDragAndDrop

 <ScheduleComponent
 allowDragAndDrop={true}>
</ScheduleComponent> |

| Enabling inline editing of appointments | **Property:** *allowInline*

 <EJ.Schedule
 id="Schedule" allowInline={true}>
 </EJ.Schedule> | Not applicable |

| Allow keyboard interactions | **Property:** *allowKeyboardNavigation*

 <EJ.Schedule
 id="Schedule" allowKeyboardNavigation={true}>
 </EJ.Schedule> | **Property:**
allowKeyboardInteraction

 <ScheduleComponent
 allowKeyboardInteraction={true}>
</ScheduleComponent> |

| Enable resizing of appointments | **Property:** *enableAppointmentResize*

 <EJ.Schedule
 id="Schedule" enableAppointmentResize={true}>
 </EJ.Schedule> | **Property:** *allowResizing*

 <ScheduleComponent allowResizing={true}>
</ScheduleComponent> |

| Blocking time intervals | **Property:** *blockoutSettings*

 var blockData = {
enable:
 true,
dataSource: [{
BlockId: 101,
BlockStartTime: new Date(2014, 4, 5, 10,
 00),
BlockEndTime: new Date(2014, 4, 5, 11, 00),
BlockSubject:
 "Service",
IsBlockAppointment: true
}],
id: "BlockId",
startTime:
 "BlockStartTime",
endTime: "BlockEndTime",
subject:
 "BlockSubject",
isBlockAppointment: "IsBlockAppointment"
};
var Scheduler =
 React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule"
 currentDate={new Date(2017, 5, 5)}
 blockoutSettings={blockData}>
</EJ.Schedule>
);

}); | Not applicable |

| Categorizing the appointments | **Property:** *categorizeSettings*

 var dataManager =
 [{
Id: 1,
Subject: "Testing",
StartTime: new Date("2015/11/7 09:00
 AM"),
EndTime: new Date("2015/11/7 10:00 AM"),
categorize: "1",
});
var
 categorizeData = {
enable: true,
allowMultiple: true,
dataSource: [
{ text: "Blue
 Category", id: 1, color: "#43b496", fontColor: "#ffffff" }],
text: "text",id: "id",color:
 "color"fontColor: "fontColor"
};
var Scheduler = React.createClass({
render: function
 () {
return (
<EJ.Schedule id="Schedule" currentDate={new Date(2017, 5, 5)}
 categorizeSettings={categorizeData} appointmentSettings-
 dataSource={dataManager}>
</EJ.Schedule>
);

}); | Not applicable |

| Setting cell height | **Property:** *cellHeight*

 <EJ.Schedule id="Schedule" cellHeight
 ="30px">
</EJ.Schedule> | Not applicable |

| Cell template | **Property:** *workCellsTemplateId*

 <EJ.Schedule id="Schedule"
allDayCellsTemplateId="#allDayTemplate"
workCellsTemplateId="#workTemplate">
</EJ.Schedule> | **Property:** *cellTemplate*

<ScheduleComponent
cellTemplate={this.cellTemplate.bind(this)}>
</ScheduleComponent> |

| Setting cell width | **Property:** *cellWidth*

 <EJ.Schedule id="Schedule"
cellWidth="30px">
</EJ.Schedule> | Not applicable |

| CSS class | **Property:** *cssClass*

 <EJ.Schedule id="Schedule"
cssClass="customStyle">
 </EJ.Schedule>
 | **Property:** *cssClass*

<ScheduleComponent cssClass= 'customStyle'>
</ScheduleComponent> |

| Enabling Context-menu option | **Property:** *contextMenuSettings*

 var contextMenuData =
{
enable: true,
menuItems: {
appointment: [
{ id: "open", text: "Open
Appointment" },
{ id: "delete", text: "Delete Appointment" }
]
,
var
Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule
id="Schedule" currentDate={new Date(2017, 5, 5)}
contextMenuSettings={contextMenuData}>
</EJ.Schedule>
);
}
}); | Not applicable
|

| Current view | **Property:** *currentView*

 <EJ.Schedule id="Schedule"
currentView={ej.Schedule.CurrentView.Workweek}>
</EJ.Schedule> | **Property:** *currentView*

 <ScheduleComponent currentView= "WorkWeek">
</ScheduleComponent> |

| Date format | **Property:** *dateFormat*

 <EJ.Schedule id="Schedule"
dateFormat="yyyy/MM/dd">
</EJ.Schedule> | **Property:** *dateFormat*

<ScheduleComponent dateFormat= "yyyy/MM/dd">
</ScheduleComponent> |

| Date header template | **Property:** *dateHeaderTemplateId*

 <EJ.Schedule id="Schedule"
dateHeaderTemplateId="#dateTemplate">
</EJ.Schedule> | **Property:** *dateHeaderTemplate*

 <ScheduleComponent
dateHeaderTemplate={this.dateHeaderTemplate.bind(this)}>
</ScheduleComponent> |

| Editor template | Not Applicable | **Property:** *editorTemplate*

 <ScheduleComponent
editorTemplate={this.editorTemplate.bind(this)}>
</ScheduleComponent> |

| Enable load on demand | **Property:** *enableLoadOnDemand*

 <EJ.Schedule id="Schedule"
enableLoadOnDemand= {true}>
</EJ.Schedule> | Not applicable |

| Enable persistence | **Property:** *enablePersistence*

 <EJ.Schedule id="Schedule"
enablePersistence= {true}>
</EJ.Schedule> | **Property:** *enablePersistence*

<ScheduleComponent enablePersistence= {true}>
</ScheduleComponent> |

| Enable RTL | **Property:** *enableRTL*

 <EJ.Schedule id="Schedule" enableRTL=
{true}>
</EJ.Schedule> | **Property:** *enablePersistence*

 <ScheduleComponent
enableRTL= {true}>
</ScheduleComponent> |

| Setting end hour of the scheduler | **Property:** *endHour* `

 <EJ.Schedule id="Schedule" endHour={18}>
</EJ.Schedule>` | **Property:** *endHour* `

 <ScheduleComponent endHour= '20:00'>
</ScheduleComponent>` |

| Setting first day of the week | **Property:** *firstDayOfWeek* `

 <EJ.Schedule id="Schedule" firstDayOfWeek={ej.Schedule.FirstDayOfWeek.Tuesday}>
</EJ.Schedule>` | **Property:** *firstDayOfWeek* `

 <ScheduleComponent firstDayOfWeek={1}>
</ScheduleComponent>` |

| Height of the scheduler | **Property:** *height* `

 <EJ.Schedule id="Schedule" height="550px">
</EJ.Schedule>` | **Property:** *height* `

 <ScheduleComponent height="550px">
</ScheduleComponent>` |

| Locale | **Property:** *locale* `

 <EJ.Schedule id="Schedule" locale="fr-CH">
</EJ.Schedule>
` | **Property:** *locale* `

 <ScheduleComponent locale='fr-CH'>
</ScheduleComponent>` |

| Priority settings for appointments | **Property:** *PrioritySettings* `

 var priorityData = {
enable: true,
dataSource: [
{ text: "None", value: "none" },
{ text: "Low", value: "low" }],
text: "text",
value: "value"
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule" prioritySettings={priorityData}>
</EJ.Schedule>
);
}
});` | Not applicable |

| Read only | **Property:** *readOnly* `

 <EJ.Schedule id="Schedule" width="100%" height="550px" readOnly={true}>
</EJ.Schedule>
` | **Property:** *readonly* `

 <ScheduleComponent readonly={true}>
</ScheduleComponent>` |

| Reminder settings | **Property:** *reminderSettings* `

 var reminderData = {
enable: true,
alertBefore: 10
};
<EJ.Schedule id="Schedule" reminderSettings={reminderData}>
</EJ.Schedule>` | Not applicable |

| Resource header template | **Property:** *resourceHeaderTemplateId* `

 var groupData = {
resources: ["Rooms"] };
var roomData = {
dataSource: [
{ ResourceText: "ROOM1", id: 1, ResourceColor: "orange" }],
text: "ResourceText", id: "id", color: "ResourceColor"};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule" group={groupData} resourceHeaderTemplateId="#resTemplate" appointmentSettings- resourceFields="RoomId">
<resources>
<resource allowMultiple={false} field="RoomId" title="Room" name="Rooms" resourceSettings={roomData}>
</resource>
</resources>
</EJ.Schedule>
);
}
});` | **Property:** *resourceHeaderTemplate* `

 private projectData: Object[] = [
{ text: 'PROJECT 1', id: 1, color: '#cb6bb2' }];
<ScheduleComponent group={ { resources: ['Projects'] } } resourceHeaderTemplate={this.resourceHeaderTemplate.bind(this)}>
<ResourcesDirective>
<ResourceDirective field='ProjectId' title='Choose Project' name='Projects' dataSource={this.projectData} textField='text' idField='id' colorField='color'>
</ResourceDirective>
</ResourcesDirective>
</ScheduleComponent>` |

| Current date of the scheduler | **Property:** *currentDate*

 <EJ.Schedule id="Schedule" currentDate={new Date(2017, 5, 5)}>
</EJ.Schedule>
 | **Property:** *selectedDate*

 <ScheduleComponent selectedDate={new Date(2018, 5, 5)}>
</ScheduleComponent> |

| Show all day row | **Property:** *showAllDayRow*

 <EJ.Schedule id="Schedule" width="100%" height="550px" showAllDayRow={false}>
</EJ.Schedule> | Not applicable |

| Show appointment navigator | **Property:** *showAppointmentNavigator*

 <EJ.Schedule id="Schedule" showAppointmentNavigator={false}>
</EJ.Schedule> | Not applicable |

| Show delete confirmation dialog | **Property:** *showDeleteConfirmationDialog*

 <EJ.Schedule id="Schedule" showDeleteConfirmationDialog={false}>
</EJ.Schedule> | Not applicable |

| Show header bar | **Property:** *showHeaderBar*

 <EJ.Schedule id="Schedule" showHeaderBar={false}>
</EJ.Schedule> | **Property:** *showHeaderBar*

 <ScheduleComponent showHeaderBar={false}>
</ScheduleComponent> |

| Show location field in event window | **Property:** *showLocationField*

 <EJ.Schedule id="Schedule" showLocationField={false}>
</EJ.Schedule> | Not applicable |

| Show time zone fields in event window | **Property:** *showTimeZoneFields*

 <EJ.Schedule id="Schedule" showTimeZoneFields={false}>
</EJ.Schedule> | Not applicable |

| Show previous and next month dates in month view | **Property:** *showNextPrevMonth*

 <EJ.Schedule id="Schedule" showNextPrevMonth={false}>
</EJ.Schedule> | Not applicable |

| Show overflow button | **Property:** *showOverflowButton*

 <EJ.Schedule id="Schedule" showOverflowButton={false}>
</EJ.Schedule> | Not applicable |

| Show quick popup | **Property:** *showQuickWindow*

 <EJ.Schedule id="Schedule" showQuickWindow={false}>
</EJ.Schedule> | **Property:** *showQuickInfo*

 <ScheduleComponent showQuickInfo={false}>
</ScheduleComponent> |

| Show current time indicator | **Property:** *showCurrentTimeIndicator*

 <EJ.Schedule id="Schedule" showCurrentTimeIndicator={false}>
</EJ.Schedule> | **Property:** *showTimeIndicator*

 <ScheduleComponent showTimeIndicator={false}>
</ScheduleComponent> |

| Show week number | Not Applicable | **Property:** *showWeekNumber*

 <ScheduleComponent showWeekNumber={false}>
</ScheduleComponent> |

| Show weekend days | **Property:** *showWeekend*

 <EJ.Schedule id="Schedule" showWeekend={false}>
</EJ.Schedule> | **Property:** *showWeekend*

 <ScheduleComponent showWeekend={false}>
</ScheduleComponent> |

| Setting start hour of the scheduler | **Property:** *startHour*

 <EJ.Schedule id="Schedule" startHour={7}>
</EJ.Schedule> | **Property:** *startHour*

 <ScheduleComponent startHour= '15:00'>
</ScheduleComponent> |

| Setting time mode on scheduler | **Property:** *timeMode*

 <EJ.Schedule id="Schedule" timeMode={ej.Schedule.TimeMode.Hour24}>
</EJ.Schedule> | Not applicable |

| Setting timezone for scheduler | **Property:** *timeZone*

 <EJ.Schedule id="Schedule" timeZone="UTC +05:30">
</EJ.Schedule> | **Property:** *timezone*

 <ScheduleComponent timeZone='UTC'>
</ScheduleComponent> |

| Views in scheduler | **Property:** *views*

 <EJ.Schedule id="Schedule" views=[["Day", "Week", "WorkWeek", "Month", "CustomView"]]>
</EJ.Schedule> | **Property:** *views*

 <ScheduleComponent>
<ViewsDirective>
<ViewDirective option='Day' />
<ViewDirective option='Week' />
<ViewDirective option='WorkWeek' />
<ViewDirective option='Month' />
</ViewsDirective>
</ScheduleComponent> |

| Width of the scheduler | **Property:** *width*

 <EJ.Schedule id="Schedule" width="100%">
</EJ.Schedule> | **Property:** *width*

 <ScheduleComponent width="100%">
</ScheduleComponent> |

| Working days | **Property:** *workWeek*

 <EJ.Schedule id="Schedule" workWeek=[["Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]]>
</EJ.Schedule> | **Property:** *workDays*

 <ScheduleComponent workDays= {[1, 2, 4, 5]}>
</ScheduleComponent> |

| Working hours | **Property:** *workHours*

 var workHourData = {
highlight: true,
start: 8,
end: 16
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule" width="100%" height="525px" workHours={workHourData} currentDate={new Date(2017, 5, 5)}>
</EJ.Schedule>
);

}); | **Property:** *workHours*

 <ScheduleComponent workHours= { {
highlight: true, start: '11:00', end: '20:00' } }>
</ScheduleComponent> |

Resources

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| To define resource datasource | **Property:** *resources*

var roomData = {
dataSource: [
{ ResourceText: "ROOM1", id: 1, ResourceColor: "orange" }],
text: "ResourceText", id: "id", color: "ResourceColor"};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule" appointmentSettings- resourceFields="RoomId">
<resources>
<resource allowMultiple={false} field="RoomId" title="Room" name="Rooms" resourceSettings={roomData}>
</resource>
</resources>
</EJ.Schedule>
);

}); | **Property:** *resources*

 private projectData: Object[] = [
{ text: 'PROJECT 1', id: 1, color: '#cb6bb2' }];
<ScheduleComponent>
<ResourcesDirective>
<ResourceDirective field='ProjectId' title='Choose Project' name='Projects' dataSource={this.projectData} textField='text' idField='id' colorField='color'>
</ResourceDirective>
</ResourcesDirective>
</ScheduleComponent> |

| Allowing multiple selection of resources in event window | **Property:** *allowMultiple*

 var groupData = { resources: ["Rooms, Owners"] };
var roomData = {
dataSource: [
{ text: "ROOM 1", id: 1, groupId: 1, color: "#cb6bb2" },
{ text: "ROOM 2", id: 2, groupId: 1, color:

```

"#56ca85" ]],<br>text: "text", id: "id", groupId: "groupId", color: "color");<br>var ownerData =
{<br>dataSource: [<br>{ text: "Nancy", id: 1, groupId: 1, color: "#ffaa00" },<br>{ text: "Steven",
id: 3, groupId: 2, color: "#f8a398"}],<br>text: "text", id: "id", groupId: "groupId", color:
"color");<br>var Scheduler = React.createClass({<br>render: function () {<br>return
(<br><EJ.Schedule id="Schedule" group={groupData} appointmentSettings-resourceFields=
"RoomId,OwnerId"><br><resources><br><resource allowMultiple={false} field="RoomId"
title="Room" name="Rooms" resourceSettings={roomData}><br></resource><br><resource
allowMultiple={true} field="OwnerId" title="Owner" name="Owners"
resourceSettings={ownerData}><br></resource><br></resources><br></EJ.Schedule><br>);<br>
<br>}); | Property: allowMultiple <br><br> private projectData: Object[] = [<br>{ text: 'PROJECT
1', id: 1, color: '#cb6bb2'},<br>{ text: 'PROJECT 2', id: 2, color: '#df5286'}];<br>private
categoryData: Object[] = [<br>{ text: 'Category 1', id: 1, color: '#cb6bb2' }<br>{ text: 'Category 2',
id: 2, color: '#df5286'}];<br><ScheduleComponent group={ { resources: ['Projects', 'Categories']
} }><br><ResourcesDirective><br><ResourceDirective field='ProjectId' title='Choose Project'
name='Projects' allowMultiple={true} dataSource={this.projectData} textField='text' idField='id'
colorField='color'><br></ResourceDirective><br><ResourceDirective field='CategoryId'
title='Choose Category' name='Categories' allowMultiple={true}
dataSource={this.categoryData} textField='text' idField='id'
colorField='color'><br></ResourceDirective><br></ResourcesDirective><br></ScheduleCompon
ent> |

```

```

| Setting different work hours for each resource | <br><br> var groupData = {<br>resources:
["Rooms"] };<br>var roomData = {<br>dataSource: [<br>{ text: "ROOM 1", id: 1, groupId: 1,
color: "#cb6bb2", on: 10, off: 18, customDays: ["monday", "wednesday", "friday"]
}<br>,<br>text: "text", id: "id", groupId: "groupId", color: "color", start: "on", end: "off",
workWeek: "customDays"}];<br>var Scheduler = React.createClass({<br>render: function ()
{<br>return (<br><EJ.Schedule id="Schedule" group={groupData} appointmentSettings-
resourceFields= "RoomId"><br><resources><br><resource allowMultiple={false} field="RoomId"
title="Room" name="Rooms"
resourceSettings={roomData}><br></resource><br></resources><br></EJ.Schedule><br>);<br>
<br>}); | private projectData: Object[] = [<br>{ text: 'PROJECT 1', id: 1, color: '#cb6bb2', ,
workDays: [1, 3, 5], startHour: '08:00', endHour: '17:00' }];<br><ScheduleComponent group={ {
resources: ['Projects'] } }><br><ResourcesDirective><br><ResourceDirective field='ProjectId'
title='Choose Project' name='Projects' allowMultiple={true} dataSource={this.projectData}
textField='text' idField='id' colorField='color' workDaysField='workDays'
startHourField='startHour'
endHourField='endHour'><br></ResourceDirective><br></ResourcesDirective><br></ScheduleC
omponent> |

```

Group

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| To group the resources in scheduler layout | **Property:** group

 var groupData =
{
resources: ["Rooms"] };
var roomData = {
dataSource: [
{ ResourceText:


```

"ROOM1", id: 1, ResourceColor: "orange" }],<br>text: "ResourceText", id: "id", color:
"ResourceColor");<br>var Scheduler = React.createClass({<br>render: function () {<br>return
<br><EJ.Schedule id="Schedule" group={groupData} appointmentSettings-
resourceFields="RoomId"><br><resources><br><resource allowMultiple={false} field="RoomId"
title="Room" name="Rooms"
resourceSettings={roomData}><br></resource><br></resources><br></EJ.Schedule><br>}<br>}); | Property: group <br><br> private projectData: Object[] = [<br>{ text: 'PROJECT 1', id: 1,
color: '#cb6bb2' }];<br><ScheduleComponent group={ { resources: ['Projects'] }
}><br><ResourcesDirective><br><ResourceDirective field='ProjectId' title='Choose Project'
name='Projects' dataSource={this.projectData} textField='text' idField='id'
colorField='color'><br></ResourceDirective><br></ResourcesDirective><br></ScheduleCompon
ent> |

```

```

| Allow group editing | Property: allowGroupEditing <br><br> var groupData = {<br>resources:
["Rooms"],<br>allowGroupEdit: true };<br>var roomData = {<br>dataSource: [<br>{
ResourceText: "ROOM1", id: 1, ResourceColor: "orange" }],<br>text: "ResourceText", id: "id",
color: "ResourceColor");<br>var Scheduler = React.createClass({<br>render: function ()
{<br>return (<br><EJ.Schedule id="Schedule" group={groupData} appointmentSettings-
resourceFields="RoomId"><br><resources><br><resource allowMultiple={false} field="RoomId"
title="Room" name="Rooms"
resourceSettings={roomData}><br></resource><br></resources><br></EJ.Schedule><br>}<br>}); | Property: allowGroupEdit <br><br> private projectData: Object[] = [<br>{ text: 'PROJECT
1', id: 1, color: '#cb6bb2' }];<br><ScheduleComponent group={ { allowGroupEdit: true,
resources: ['Projects'] } }><br><ResourcesDirective><br><ResourceDirective field='ProjectId'
title='Choose Project' name='Projects' dataSource={this.projectData} textField='text'
idField='id'
colorField='color'><br></ResourceDirective><br></ResourcesDirective><br></ScheduleCompon
ent> |

```

```

| Grouping resources by date | Not applicable | Property: byDate <br><br> private projectData:
Object[] = [<br>{ text: 'PROJECT 1', id: 1, color: '#cb6bb2' }];<br><ScheduleComponent group={ {
byDate: true, resources: ['Projects'] } }><br><ResourcesDirective><br><ResourceDirective
field='ProjectId' title='Choose Project' name='Projects' dataSource={this.projectData}
textField='text' idField='id'
colorField='color'><br></ResourceDirective><br></ResourcesDirective><br></ScheduleCompon
ent> |

```

```

| Grouping resources based on its group ID | Not applicable | Property: byGroupId <br><br> private
projectData: Object[] = [<br>{ text: 'PROJECT 1', id: 1, color: '#cb6bb2'},<br>{ text: 'PROJECT 2',
id: 2, color: '#56ca85'}];<br>private categoryData: Object[] = [<br>{ text: 'Development', id: 1,
groupId: 1, color: '#7fa900' },<br>{ text: 'Testing', id: 2, groupId: 2, color: '#56ca85' },<br>{ text:
'Documentation', id: 3, groupId: 2, color: '#7499e1' }];<br><ScheduleComponent group={ {
byGroupId: true, resources: ['Projects', 'Categories'] } }
><br><ResourcesDirective><br><ResourceDirective field='ProjectId' title='Choose Project'
name='Projects' dataSource={this.projectData} textField='text' idField='id' groupIdField:
'groupId' colorField='color'><br></ResourceDirective><br><ResourceDirective field='CategoryId'

```

```
title='Category' name='Categories' allowMultiple={true} dataSource={this.categoryData}
textField='text' idField='id' groupIDField: 'groupId'
colorField='color'><br></ResourceDirective><br></ResourcesDirective><br></ScheduleComponent> |
```

```
| Enabling compact view on mobile mode | Not applicable | Property: enableCompactView <br><br>
private projectData: Object[] = [<br>{ text: 'PROJECT 1', id: 1, color: '#cb6bb2'
}];<br><ScheduleComponent group={ { enableCompactView: false, resources: ['Projects'] }
}><br><ResourcesDirective><br><ResourceDirective field='ProjectId' title='Choose Project'
name='Projects' dataSource={this.projectData} textField='text' idField='id'
colorField='color'><br></ResourceDirective><br></ResourcesDirective><br></ScheduleComponent> |
```

```
| Header tooltip template | Not applicable | Property: headerTooltipTemplate <br><br> private
projectData: Object[] = [<br>{ text: 'PROJECT 1', id: 1, color: '#cb6bb2'
}];<br><ScheduleComponent headerTooltipTemplate={this.resourceHeaderTemplate.bind(this)}
group={ { resources: ['Projects'] } }><br><ResourcesDirective><br><ResourceDirective
field='ProjectId' title='Choose Project' name='Projects' dataSource={this.projectData}
textField='text' idField='id'
colorField='color'><br></ResourceDirective><br></ResourcesDirective><br></ScheduleComponent> |
```

Header Rows

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

```
| Adding custom rows in the header in timeline views | Not applicable | Property: HeaderRows
<br><br> <ScheduleComponent><br> <HeaderRowsDirective><br><HeaderRowDirective
option='Month' template={this.monthTemplate.bind(this)} /><br><HeaderRowDirective
option='Week' template={this.weekTemplate.bind(this)} /><br><HeaderRowDirective
option='Date' /><br></HeaderRowsDirective><br>< ViewsDirective ><br><ViewDirective
option='TimelineMonth' interval={12} /><br></ViewsDirective><br></ScheduleComponent> |
```

TimeScale

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

```
| Enabling time scale | Property: enable <br><br> var timeScaleData = {<br>enable: true };<br>var
Scheduler = React.createClass({<br>render: function () {<br>return (<br><EJ.Schedule
id="Schedule" timeScale={timeScaleData}><br></EJ.Schedule><br>);<br>}<br>}); | Property:
enable <br><br> <ScheduleComponent timeScale={ { enable: true }
}><br></ScheduleComponent> |
```

```
| Setting major interval on time scale | Property: majorSlot <br><br> var timeScaleData =
{<br>enable: true, majorSlot: 60 };<br>var Scheduler = React.createClass({<br>render: function ()
{<br>return (<br><EJ.Schedule id="Schedule" timeScale={timeScaleData}
><br></EJ.Schedule><br>);<br>}<br>}); | Property: interval <br><br> <ScheduleComponent
timeScale={ { enable: true, interval: 60 } }><br></ScheduleComponent> |
```

| Setting slot count on time scale | **Property:** *minorSlotCount*

 var timeScaleData = {
enable: true, majorSlot: 60, minorSlotCount: 6 };
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule" timeScale={timeScaleData}>
</EJ.Schedule>
);
}
}); | **Property:** *slotCount*

 <ScheduleComponent timeScale={ { enable: true, interval: 60, slotCount: 6 } }>
</ScheduleComponent> |

| Defining major slot template | **Property:** *majorSlotTemplateId*

 var timeScaleData = {
enable: true, majorSlot: 60, minorSlotCount: 6, majorSlotTemplateId: "#majorTemplate" };
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule" timeScale={timeScaleData}>
</EJ.Schedule>
);
}
}); | **Property:** *majorSlotTemplate*

 <ScheduleComponent timeScale={ { enable: true, interval: 60, slotCount: 6, majorSlotTemplate:this.majorSlotTemplate.bind(this) } }>
</ScheduleComponent> |

| Defining minor slot template | **Property:** *minorSlotTemplateId*

 var timeScaleData = {
enable: true, majorSlot: 60, minorSlotCount: 6, minorSlotTemplateId: "#minorTemplate" };
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule" timeScale={timeScaleData}>
</EJ.Schedule>
);
}
}); | **Property:** *minorSlotTemplate*

 <ScheduleComponent timeScale={ { enable: true, interval: 60, slotCount: 6, minorSlotTemplate:this.minorSlotTemplate.bind(this) } }>
</ScheduleComponent> |

Quick info templates

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| Template for quick popup | Not applicable | **Property:** *quickInfoTemplates*

 <ScheduleComponent quickInfoTemplates: {
{
header:this.headerTemplate.bind(this),
content:this.contentTemplate.bind(this),
footer:this.footerTemplate.bind(this)
} }></ScheduleComponent> |

Event settings

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| Datasource for events | **Property:** *dataSource*

 var dataManager = [{
Id: 1,
Subject: "Development",
StartTime: new Date(2018, 1, 15, 10, 0),
EndTime: new Date(2018, 1, 15, 12, 30)
}];
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule" currentDate={new Date(2017, 5, 5)} appointmentSettings-dataSource={dataManager}>
</EJ.Schedule>
);
}
}); | **Property:** *dataSource*

 private data: object [] = [{
Id: 1,
Subject: 'Paris',
StartTime: new Date(2018, 1, 15, 10, 0),
EndTime: new Date(2018, 1, 15, 12, 30),
IsAllDay: false,
RecurrenceID: 10
}];
<ScheduleComponent eventSettings={ { dataSource: this.data } }>
</ScheduleComponent> |

| Appointment fields | `<EJ.Schedule id="Schedule" currentDate={new Date(2017, 5, 5)} appointmentSettings-datasource={
id: "Id",
subject: "Subject",
startTime: "StartTime",
endTime: "EndTime",
description: "Description",
allDay: "AllDay",
recurrence: "Recurrence",
recurrenceRule: "RecurrenceRule">
</EJ.Schedule>` | **Property:** `eventsSettings` `

` private data: object
`[] =`
`[{
Id: 1,
Subject: 'Paris',
StartTime: new Date(2018, 1, 15, 10, 0),
EndTime: new Date(2018, 1, 15, 12, 30),
IsAllDay: false,
RecurrenceID: 10
}}];
<ScheduleComponent eventSettings={ { datasource: this.data,
fields: {
id: 'Id',
subject: { name: 'Subject' },
isAllDay: { name: 'IsAllDay' },
location: { name: 'Location' },
description: { name: 'Description' },
startTime: { name: 'StartTime' },
endTime: { name: 'EndTime' },
recurrenceID : { name: 'RecurrenceID' }

 }>
</ScheduleComponent>` |

| Enabling tooltip for appointments | **Property:** `enable` `

` var tooltipData = {
enable: true};
var Scheduler = React.createClass({
render: function () {
return
`
<EJ.Schedule id="Schedule"`
`tooltipSettings={tooltipData}>
</EJ.Schedule>
};

}); | Property: enableTooltip

 <ScheduleComponent eventSettings={ {datasource: this.data, enableTooltip: true}
>
</ScheduleComponent> |`

| Tooltip template for appointments | **Property:** `templateId` `

` var tooltipData = {
enable: true,templateId: "#tooltipTemplate"};
var Scheduler =
`React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule"`
`tooltipSettings={tooltipData}>
</EJ.Schedule>
};

});
Note: Here tooltip`
`setting for events is maintained separately | Property: tooltipTemplate

`
`<ScheduleComponent eventSettings={ {datasource: this.data, enableTooltip: true,`
`tooltipTemplate: this.template.bind(this)} >
</ScheduleComponent>` |

| Template for appointments | **Property:** `appointmentTemplateId` `

` `<EJ.Schedule`
`id="Schedule" appointmentTemplateId="#appTemplate">
</EJ.Schedule>` | **Property:**
`template` `

` `<ScheduleComponent eventSettings={ {datasource: this.data,`
`template:this.eventTemplate.bind(this)} >
</ScheduleComponent>` |

| Query | **Property:** `query` `

` var dataManager = ej.DataManager({
url:
`"http://mvc.syncfusion.com/OdataServices/Northwnd.svc/"
});
var queryManager =`
`ej.Query().from("Events").take(10);
var Scheduler = React.createClass({
render: function`
`() {
return (
<EJ.Schedule id="Schedule" appointmentSettings-`
`datasource={dataManager} appointmentSettings-`
`query={queryManager}>
</EJ.Schedule>
};

}); | Property: query

 private
dataManger: DataManager = new DataManager({
url:
'https://ej2services.syncfusion.com/production/web-services/api/Schedule',
adaptor: new
ODataAdaptor
});
<ScheduleComponent eventSettings={ { datasource: this.dataManger,
query: new Query().addParams('ej2schedule', 'true') } >
</ScheduleComponent> |`

| Define which resource level color applied to events | Not applicable | **Property:** `resourceColorField`
`

` public roomData: Object[] = [`
`{ RoomText: 'ROOM 1', Id: 1, RoomGroupld: 1,

```
RoomColor: '#cb6bb2' }<br>];<br>public ownerData: Object[] = [<br>{ OwnerText: 'OWNER 1',
Id: 1, OwnerGroupId: 1, OwnerColor: '#cb6bb2' }<br>];<br><ScheduleComponent
eventSettings= { { resourceColorField: 'Owners' } }<br>group= { { resources: ['Rooms', 'Owners']
} }><br><ResourcesDirective><br><ResourceDirective field='RoomId' title='Room'
name='Rooms' allowMultiple={false} dataSource={this.roomData} textField='RoomText'
idField='Id' groupIDField= 'RoomGroupId'
colorField='RoomColor'><br></ResourceDirective><br><ResourceDirective field='OwnerId'
title='Owner' name='Owners' allowMultiple={true} dataSource={this.ownerData}
textField='OwnerText' idField='Id' groupIDField= 'OwnerGroupId'
colorField='OwnerColor'><br></ResourceDirective><br></ResourcesDirective><br></ScheduleC
omponent> |
```

Methods

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| To add appointments manually | **Method:** *saveAppointment()*

 <EJ.Schedule
dataSource={window.scheduleData}>
</EJ.Schedule>
 var data = {
 Id: 1,
Subject:
"Testing",
StartTime: new Date(2014, 4, 5, 10, 00),
EndTime: new Date(2014, 4, 5, 12,
00)};
 var scheduleobj =
\$("#Schedule").data("ejSchedule");
scheduleobj.saveAppointment(data); | **Method:**
addEvent()

 <ScheduleComponent ref={schedule => this.scheduleObj =
schedule}>
</ScheduleComponent>
public data: Object[]=
{
Id: 1,
Subject: 'Testing',
StartTime: new Date(2018, 1, 11, 10, 0),
EndTime: new
Date(2018, 1, 11, 11, 0),
IsAllDay: false};
this.scheduleObj.addEvent(data);
}|

| To add resources dynamically | **Method:** *addResource()*

 <EJ.Schedule
dataSource={window.scheduleData}>
</EJ.Schedule>
 var data = { text: "Paul", id: 1,
groupId: 3, color: "#cc99ff" };
 var index = 0;
 var scheduleobj =
\$("#Schedule").data("ejSchedule");
 scheduleobj.addResource(data, "Owners", index); |
Method: *addResource()*

 <ScheduleComponent ref={schedule => this.scheduleObj =
schedule}>
</ScheduleComponent>
public data: Object[]={ text: "Paul", id: 1, groupId: 3,
color: "#cc99ff" };
 public index = 0;
this.scheduleobj.addResource(data, "Owners",
index); |

| databind | Not applicable | **Method:** *dataBind()*

 <ScheduleComponent ref={schedule
=> this.scheduleObj = schedule}>
</ScheduleComponent>
this.scheduleObj.dataBind() |

| Delete appointment manually | **Method:** *deleteAppointment()*

 <EJ.Schedule
dataSource={window.scheduleData}>
</EJ.Schedule>
 var data = {
 Id: 1,
Subject:
"Testing",
StartTime: new Date(2014, 4, 5, 10, 00),
EndTime: new Date(2014, 4, 5, 12,
00)};
 var scheduleobj = \$("#Schedule").data("ejSchedule");

scheduleobj.deleteAppointment(data); | **Method:** *deleteEvent()*

<ScheduleComponent ref={schedule => this.scheduleObj =
schedule}>
</ScheduleComponent>
public data: Object[] = {
Id: 1,
Subject:

```

“Testing”,<br>StartTime: new Date(2014, 4, 5, 10, 00),<br>EndTime: new Date(2014, 4, 5, 12, 00));<br>this.scheduleobj.deleteEvent(data); |

```

```

| destroy scheduler | Method: destroy() <br><br> <EJ.Schedule
dataSource={window.scheduleData}><br></EJ.Schedule><br> var scheduleobj =
$("#Schedule").data("ejSchedule"); <br> schdeuleObj.destroy(); | Method: destroy() <br><br>
<ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>this.scheduleObj.destroy() |

```

```

| Get cell details | Method: getSlotByElement() <br><br> <EJ.Schedule
dataSource={window.scheduleData}><br></EJ.Schedule><br> var scheduleobj =
$("#Schedule").data("ejSchedule"); <br> var $td = $(".e-draggableworkarea table tr
td").first();<br>var slotDetails =scheduleObj.getSlotByElement($td); | Method: getCellDetails()
<br><br> <ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>private td = document.querySelector(".e-work-
cells"); <br>private cellDetail = this.scheduleobj.getCellDetails(td); |

```

```

| Get current view appointments | Method: getCurrentViewAppointments() <br><br> <EJ.Schedule
dataSource={window.scheduleData}> <br> </EJ.Schedule><br>var scheduleobj =
$("#Schedule").data("ejSchedule"); <br> var currApp=
scheduleobj.getCurrentViewAppointments(); | Method: getCurrentViewEvents() <br><br>
<ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>private currApp=
this.scheduleobj.getCurrentViewEvents(); |

```

```

| Get entire appointment collection | Method: getAppointments() <br><br> <EJ.Schedule
dataSource={window.scheduleData}><br></EJ.Schedule><br> var scheduleobj =
$("#Schedule").data("ejSchedule");<br>var AppDetails = scheduleobj.getAppointments(); |
Method: getEvents() <br><br> <ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>public AppDetails= this.scheduleobj.getEvents(); |

```

```

| Get current view dates | Not applicable | Method: getCurrentViewDates() <br><br>
<ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>public AppDetails =
this.scheduleobj.getCurrentViewDates(); |

```

```

| Get event details | Not applicable | Method: getEventDetails() <br><br> public AppDetails =
scheduleobj.getEventDetails(appElement);<br><ScheduleComponent ref={schedule =>
this.scheduleObj = schedule}><br></ScheduleComponent> |

```

```

| Get occurrences using event ID | Not applicable | Method: getOccurrencesByID() <br><br>
<ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>public recCollection =
this.scheduleobj.getOccurrencesByID(1); |

```

```

| Get occurrences in the provided date range | Not applicable | Method: getOccurrencesByRange()
<br><br> <ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>public sDate = new Date(2018, 1, 12); <br> public

```

```
eDate = new Date(2018, 1, 17); <br> public resCollection =
this.scheduleobj.getOccurrencesByRange(sDate, eDate); |

| Get resource details using index | Not applicable | Method: getResourceByIndex() <br><br>
<ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>public resCollection =
this.scheduleobj.getResourceByIndex(2); |

| Show spinner | Not applicable | Method: showSpinner() <br><br> <ScheduleComponent
ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>this.scheduleObj.showSpinner() |

| Hide spinner | Not applicable | Method: hideSpinner() <br><br> <ScheduleComponent
ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>this.scheduleObj.hideSpinner() |

| Check whether the time slot is available | Not applicable | Method: isSlotAvalible() <br><br>
<ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>public sTime = new Date(2018, 1, 14, 09, 30); <br>
public etime = new Date(2018, 1, 14, 10, 30); <br> public resCollection =
this.scheduleobj.isSlotAvalible(sTime, eTime); |

| Open the event window manually | Not applicable | Method: openEditor() <br><br>
<ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>public td = document.querySelector(".e-content-
table tbody tr td");<br>public cellDetail = this.scheduleobj.getCellDetails(td);<br>
this.scheduleobj.openEditor(cellDetail); |

| Refresh the scheduler | Method: refresh() <br><br> <EJ.Schedule
dataSource={window.scheduleData}><br></EJ.Schedule><br> var scheduleobj =
$("#Schedule").data("ejSchedule"); <br> scheduleObj.refresh();| Method: refresh() <br><br>
<ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>this.scheduleObj.refresh() |

| Refresh the events | Method: refreshAppointments() <br><br> <EJ.Schedule
dataSource={window.scheduleData}><br></EJ.Schedule><br> var scheduleobj =
$("#Schedule").data("ejSchedule"); <br> scheduleObj.refreshAppointments();| Method:
refreshEvents() <br><br> <ScheduleComponent ref={schedule => this.scheduleObj =
schedule}><br></ScheduleComponent><br>this.scheduleObj.refreshEvents() |

| Refresh the scroller | Method: refreshScroller() <br><br> <EJ.Schedule
dataSource={window.scheduleData}><br></EJ.Schedule><br> var scheduleobj =
$("#Schedule").data("ejSchedule"); <br> scheduleObj.refreshScroller() | Not Applicable |

| To remove resources dynamically | Method: removeResource() <br><br> <EJ.Schedule
dataSource={window.scheduleData}><br></EJ.Schedule><br> var resID = 1;<br>var scheduleobj
= $("#Schedule").data("ejSchedule");<br>scheduleobj.removeResource(resID, "Owners");
| Method: removeResource() <br><br> <ScheduleComponent ref={schedule => this.scheduleObj =
```

```
schedule}><br></ScheduleComponent><br>public data = { text: "Paul", id: 1, groupId: 3, color:
"#cc99ff" }; <br> public index = 0; <br>this.scheduleobj.removeResource(data, "Owners");|
```

| Export schedule as PDF | **Method:** *exportSchedule()*

 <EJ.Schedule
dataSource={window.scheduleData}>
</EJ.Schedule>
 var scheduleobj =
\$("#Schedule").data("ejSchedule");

scheduleobj.exportSchedule("ActionName", "ExportToICS", null); | Not Applicable |

| Export scheduler appointments in Excel file | **Method:** *exportToExcel()*

 <EJ.Schedule
dataSource={window.scheduleData}>
</EJ.Schedule>
 var scheduleobj =
\$("#Schedule").data("ejSchedule");
 scheduleobj.exportToExcel("ActionName", null, true); |
Not Applicable |

| Print the scheduler | **Method:** *print()*

 <EJ.Schedule
dataSource={window.scheduleData}>
</EJ.Schedule>
 var scheduleobj =
\$("#Schedule").data("ejSchedule");
 scheduleObj.print(); | Not applicable |

| Filter appointments | **Method:** *filterAppointments()*

 <EJ.Schedule
dataSource={window.scheduleData}>
</EJ.Schedule>
 var scheduleobj =
\$("#Schedule").data("ejSchedule");
 var filter = [{ field: "Subject", operator: "contains",
value: "with", predicate: "or" }];
var filteredApp = scheduleobj.filterAppointments(filter); |
Not Applicable |

| Search appointments | **Method:** *searchAppointments()*

 <EJ.Schedule
dataSource={window.scheduleData}>
</EJ.Schedule>
 var scheduleobj =
\$("#Schedule").data("ejSchedule");
 var searchApp =
scheduleobj.searchAppointments("with"); | Not applicable |

| To edit appointments manually | Not applicable | **Method:** *saveEvent()*

<ScheduleComponent ref={schedule => this.scheduleObj =
schedule}>
</ScheduleComponent>
public data: Object[] = {
Id: 1,
Subject: 'Event
Edited',
StartTime: new Date(2018, 1, 11, 10, 0),
EndTime: new Date(2018, 1, 11, 11, 0),
IsAllDay: false};
this.scheduleObj.saveEvent(data);
} |

| Setting work hours | Not applicable | **Method:** *setWorkHours()*

 <ScheduleComponent
ref={schedule => this.scheduleObj =
schedule}>
</ScheduleComponent>
this.scheduleObj.setWorkHours([new Date(2017, 9,
5)], '04:00', '08:00') |

| Scrolling to specific time | Not applicable | **Method:** *scrollTo()*

 <ScheduleComponent
ref={schedule => this.scheduleObj =
schedule}>
</ScheduleComponent>
this.scheduleObj.scrollTo('12:00') |

Events

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| Fires on the beginning of each scheduler action | **Event:** *actionBegin*

 <EJ.Schedule
actionBegin={onActionBegin}>
 </EJ.Schedule>
 function onActionBegin (args){} | **Event:**

actionBegin

 <ScheduleComponent actionBegin={onActionBegin}>

</ScheduleComponent>
function onActionBegin(args){} |

| Fires on the completion of each scheduler action | **Event:** *actionComplete*

<EJ.Schedule
 actionComplete={onActionComplete}>
 </EJ.Schedule>
 function onActionComplete
 (args){} | **Event:** *actionComplete*

 <ScheduleComponent actionComplete
 ={onActionComplete}>
</ScheduleComponent>
function onActionComplete(args){} |

| Fires when the scheduler action gets failed | Not applicable | **Event:** *actionFailure*

 <ScheduleComponent actionFailure={onActionFailure}>

</ScheduleComponent>
function onActionFailure(args){} |

| Fires on appointment hover | **Event:** *appointmentHover*

 <EJ.Schedule
 appointmentHover={onAppointmentHover}>
 </EJ.Schedule>
 function
 onAppointmentHover (args){} | Not applicable |

| Fires before an appointment gets created | **Event:** *beforeAppointmentCreate*

 <EJ.Schedule beforeAppointmentCreate={onBeforeAppointmentCreate}>
 </EJ.Schedule>

 function onBeforeAppointmentCreate (args){} | Not applicable |

| Fires before an appointment gets edited | **Event:** *beforeAppointmentChange*

 <EJ.Schedule
 beforeAppointmentChange={OnBeforeAppointmentChange}>
 </EJ.Schedule>
 function
 OnBeforeAppointmentChange (args){} | Not applicable |

| Fires before an appointment gets deleted | **Event:** *beforeAppointmentRemove*

 <EJ.Schedule beforeAppointmentRemove={OnBeforeAppointmentRemove}>

 </EJ.Schedule>
 function OnBeforeAppointmentRemove (args){} | Not applicable |

| Fires before the context menu opens on scheduler | **Event:** *beforeContextMenuOpen*

 <EJ.Schedule beforeContextMenuOpen={OnBeforeContextMenuOpen}>
 </EJ.Schedule>

 function OnBeforeContextMenuOpen (args){} | Not applicable |

| Fires on cell click | **Event:** *cellClick*

 <EJ.Schedule cellClick={OnCellClick}>

 </EJ.Schedule>
 function OnCellClick (args){} | **Event:** *cellClick*

 <ScheduleComponent cellClick={OnCellClick}>
</ScheduleComponent>
function
 OnCellClick (args){} |

| Fires on cell double click | **Event:** *cellDoubleClick*

 <EJ.Schedule
 cellDoubleClick={oncellDoubleClick}>
 </EJ.Schedule>
 function oncellDoubleClick
 (args){} | **Event:** *cellClick*

 <ScheduleComponent cellDoubleClick={oncellDoubleClick}>

</ScheduleComponent>
function oncellDoubleClick (args){} |

| Fires on cell hover | **Event:** *cellHover*

 <EJ.Schedule cellHover={onCellHover }>

 </EJ.Schedule>
 function onCellHover (args){} | Not applicable |

| Fires once the scheduler is created | **Event:** *create*

 <EJ.Schedule
 create={onCreate}>
 </EJ.Schedule>
 function onCreate (args){} | **Event:** *created*

 <ScheduleComponent created={onCreated}>

</ScheduleComponent>
function onCreated (args){} |

| Fires on data binding action | Not applicable | **Event:** *dataBinding*

 <ScheduleComponent dataBinding={onDataBinding}>
</ScheduleComponent>
function
 onDataBinding(args){} |

| Fires after the data is bound to the control | Not applicable | **Event:** *dataBound*

 <ScheduleComponent dataBound={onDataBound}>
</ScheduleComponent>
function
 onDataBound(args){} |

| Fires once the scheduler is destroyed | **Event:** *destroy*

 <EJ.Schedule
 destroy={onDestroy}>
 </EJ.Schedule>
 function onDestroy (args){} | **Event:** *destroyed*

 <ScheduleComponent destroyed={onDestroyed}>

</ScheduleComponent>
function onDestroyed(args){} |

| Fires on event click | **Event:** *appointmentClick*

 <EJ.Schedule
 appointmentClick={onAppointmentClick}>
 </EJ.Schedule>
 function
 onAppointmentClick (args){} | **Event:** *eventClick*

 <ScheduleComponent eventClick
 ={onEventClick}>
</ScheduleComponent>
function onEventClick(args){} |

| Fires on event double click | **Event:** *appointmentDoubleClick*

 <EJ.Schedule
 appointmentDoubleClick={onAppointmentDoubleClick}>
 </EJ.Schedule>
 function
 onAppointmentDoubleClick (args){} | Not applicable |

| Fires for keyboard actions | **Event:** *keyDown*

 <EJ.Schedule
 keyDown={onKeyDown}>
 </EJ.Schedule>
 function onKeyDown (args){} | Not applicable
 |

| Fires on context menu item click | **Event:** *menuItemClick*

 <EJ.Schedule
 menuItemClick={onMenuItemClick}>
 </EJ.Schedule>
 function onMenuItemClick
 (args){} | Not applicable |

| Fires on navigation | **Event:** *navigation*

 <EJ.Schedule navigation={onNavigation}>

 </EJ.Schedule>
 function onNavigation (args){} | **Event:** *navigating*

 <ScheduleComponent navigating={onNavigating}>
</ScheduleComponent>
function
 onNavigating (args){} |

| Fires on popup open | **Event:** *appointmentWindowOpen*

 <EJ.Schedule
 appointmentWindowOpen={onAppointmentWindowOpen}>
 </EJ.Schedule>
 function
 onAppointmentWindowOpen (args){} | **Event:** *popupOpen*

 <ScheduleComponent
 popupOpen={onPopupOpen}>
</ScheduleComponent>
function onPopupOpen (args){}
 |

| Fires on dragging event | **Event:** *drag*

 <EJ.Schedule drag={onDrag}>
 </EJ.Schedule>

 function onDrag (args){} | **Event:** *drag*

 <ScheduleComponent drag={onDrag}>

</ScheduleComponent>
function onDrag (args){} |

| Fires on drag start | **Event:** *dragStart*

 <EJ.Schedule dragStart={onDragStart}>

 </EJ.Schedule>
 function onDragStart (args){} | **Event:** *dragStart*

 <ScheduleComponent dragStart={onDragStart}>
</ScheduleComponent>
function
 onDragStart (args){} |

| Fires on drag stop | **Event:** *dragStop*

 <EJ.Schedule dragStop={onDragStop}>
</EJ.Schedule>
 function onDragStop (args){} | **Event:** *dragStop*

 <ScheduleComponent dragStop ={onDragStop}>
</ScheduleComponent>
function onDragStop (args){} |

| Fires on overflow button click | **Event:** *overflowButtonClick*

 <EJ.Schedule overflowButtonClick={onOverflowButtonClick}>
 </EJ.Schedule>
 function onOverflowButtonClick (args){} | Not applicable |

| Fires on overflow button hover | **Event:** *overflowButtonHover*

 <EJ.Schedule overflowButtonHover={onOverflowButtonHover}>
 </EJ.Schedule>
 function onOverflowButtonHover (args){} | Not applicable |

| Fires when the reminder action takes place | **Event:** *reminder*

<EJ.Schedule reminder={onReminder}>
 </EJ.Schedule>
 function onReminder (args){} | Not applicable |

| Fires on resizing event | **Event:** *resize*

 <EJ.Schedule resize={onResize}>
</EJ.Schedule>
 function onResize (args){} | **Event:** *resizeStart*

 <ScheduleComponent resize ={onResize}>
</ScheduleComponent>
function onResize (args){} |

| Fires on resize start | **Event:** *resizeStart*

 <EJ.Schedule resizeStart={onResizeStart}>
</EJ.Schedule>
 function onResizeStart (args){} | **Event:** *resizeStart*

 <ScheduleComponent resizeStart ={onResizeStart}>
</ScheduleComponent>
function onResizeStart (args){} |

| Fires on resize stop | **Event:** *resizeStop*

 <EJ.Schedule resizeStop={onResizeStop}>
</EJ.Schedule>
 function onResizeStop (args){} | **Event:** *resizeStop*

 <ScheduleComponent resizeStop ={onResizeStop}>
</ScheduleComponent>
function onResizeStop (args){} |

| Fires on rendering of every scheduler elements | **Event:** *queryCellInfo*

 <EJ.Schedule queryCellInfo={onQueryCellInfo}>
 </EJ.Schedule>
 function onQueryCellInfo (args){} | **Event:** *renderCell*

 <ScheduleComponent renderCell ={onRenderCell}>
</ScheduleComponent>
function onRenderCell (args){} |

| Fires before the event rendering on UI | Not applicable | **Event:** *eventRendered*

 <ScheduleComponent eventRendered ={onEventRendered}>
</ScheduleComponent>
function onEventRendered (args){} |

You can refer to our [React Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [React Scheduler example](#) to know how to present and manipulate data.

How To

Add edit and remove events in React Schedule component

CRUD actions can be manually performed on appointments using `addEvent`, `saveEvent` and `deleteEvent` methods as shown below.

*Normal event***INDEX.JSX**

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef(null);
  const scheduleData = [{
    Id: 3,
    Subject: 'Testing',
    StartTime: new Date(2018, 1, 11, 9, 0),
    EndTime: new Date(2018, 1, 11, 10, 0),
    IsAllDay: false
  }, {
    Id: 4,
    Subject: 'Vacation',
    StartTime: new Date(2018, 1, 13, 9, 0),
    EndTime: new Date(2018, 1, 13, 10, 0),
    IsAllDay: false
  }];
  const eventSettings = { dataSource: scheduleData };
  const onClickAdd = () => {
    let Data = [{
      Id: 1,
      Subject: 'Conference',
      StartTime: new Date(2018, 1, 12, 9, 0),
      EndTime: new Date(2018, 1, 12, 10, 0),
      IsAllDay: false
    }, {
      Id: 2,
      Subject: 'Meeting',
      StartTime: new Date(2018, 1, 15, 10, 0),
      EndTime: new Date(2018, 1, 15, 11, 30),
      IsAllDay: false
    }];
    scheduleObj.current.addEvent(Data);
  }
  const onClickSave = () => {
    let Data = {
      Id: 3,
      Subject: 'Testing-edited',
      StartTime: new Date(2018, 1, 11, 10, 0),
      EndTime: new Date(2018, 1, 11, 11, 0),
      IsAllDay: false
    };
    scheduleObj.current.saveEvent(Data);
  }
  const onClickDelete = () => {
    scheduleObj.current.deleteEvent(4);
  }
  return (
    <div>

```

```

        <ButtonComponent id='add' title='Add'
onClick={onClickAdd}>Add</ButtonComponent>
        <ButtonComponent id='edit' title='Edit'
onClick={onClickSave}>Edit</ButtonComponent>
        <ButtonComponent id='delete' title='Delete'
onClick={onClickDelete}>Delete</ButtonComponent> <ScheduleComponent
ref={scheduleObj} width='100%' height='550px' selectedDate=
    {new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
</ScheduleComponent>
</div>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
    ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const scheduleData: Object[] = [{
        Id: 3,
        Subject: 'Testing',
        StartTime: new Date(2018, 1, 11, 9, 0),
        EndTime: new Date(2018, 1, 11, 10, 0),
        IsAllDay: false
    }, {
        Id: 4,
        Subject: 'Vacation',
        StartTime: new Date(2018, 1, 13, 9, 0),
        EndTime: new Date(2018, 1, 13, 10, 0),
        IsAllDay: false
    }];
    const eventSettings = { dataSource: scheduleData }
    const onClickAdd = (): void => {
        let Data: Object[] = [{
            Id: 1,
            Subject: 'Conference',
            StartTime: new Date(2018, 1, 12, 9, 0),
            EndTime: new Date(2018, 1, 12, 10, 0),
            IsAllDay: false
        }, {

```

```

        Id: 2,
        Subject: 'Meeting',
        StartTime: new Date(2018, 1, 15, 10, 0),
        EndTime: new Date(2018, 1, 15, 11, 30),
        IsAllDay: false
    }];
    scheduleObj.current.addEvent(Data);
}
const onClickSave = (): void => {
    let Data: Object = {
        Id: 3,
        Subject: 'Testing-edited',
        StartTime: new Date(2018, 1, 11, 10, 0),
        EndTime: new Date(2018, 1, 11, 11, 0),
        IsAllDay: false
    };
    scheduleObj.current.saveEvent(Data);
}
const onClickDelete = (): void => {
    scheduleObj.current.deleteEvent(4);
}
return (
    <div>
        <ButtonComponent id='add' title='Add'
onClick={onClickAdd}>Add</ButtonComponent>
        <ButtonComponent id='edit' title='Edit'
onClick={onClickSave}>Edit</ButtonComponent>
        <ButtonComponent id='delete' title='Delete'
onClick={onClickDelete}>Delete</ButtonComponent> <ScheduleComponent
ref={scheduleObj} width='100%' height='550px' selectedDate=
    {new Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
    </div>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Syncfusion React Schedule</title>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
        <meta name="description" content="Essential JS 2 for React
Components" />

```

```

    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdn.jsdelivr.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008c8f;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>
</head>

<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>

</html>

```

Recurrence event

INDEX.JSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DataManager, Query } from '@syncfusion/ej2-data';
const App = () => {
    const scheduleObj = useRef(null);

```

```

const scheduleData = [{
  Id: 3,
  Subject: 'Testing',
  StartTime: new Date(2018, 1, 11, 9, 0),
  EndTime: new Date(2018, 1, 11, 10, 0),
  IsAllDay: false,
  RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=3'
}, {
  Id: 4,
  Subject: 'Vacation',
  StartTime: new Date(2018, 1, 12, 11, 0),
  EndTime: new Date(2018, 1, 12, 12, 0),
  IsAllDay: false,
  RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
}];
const eventSettings = { dataSource: scheduleData }
const onClickAdd = () => {
  let Data = [{
    Id: 1,
    Subject: 'Conference',
    StartTime: new Date(2018, 1, 15, 9, 0),
    EndTime: new Date(2018, 1, 15, 10, 0),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
  }];
  scheduleObj.current.addEvent(Data);
}
const onClickSave = () => {
  let data = new
DataManager(scheduleObj.current.getCurrentViewEvents()).executeLocal(new
Query().where('RecurrenceID', 'equal', 3));
  data[0].Subject = 'Occurrence edited';
  scheduleObj.current.saveEvent(data[0], 'EditOccurrence');
}
const onClickDelete = () => {
  let Data = [{
    Id: 4,
    Subject: 'Vacation',
    RecurrenceID: 4,
    StartTime: new Date(2018, 1, 12, 11, 0),
    EndTime: new Date(2018, 1, 12, 12, 0),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
  }];
  scheduleObj.current.deleteEvent(Data, 'DeleteSeries');
}
return (
  <div>
    <ButtonComponent id='add' title='Add' onClick=
      {onClickAdd}>Add</ButtonComponent>
    <ButtonComponent id='edit' title='Edit'
onClick={onClickSave}>Edit</ButtonComponent>
    <ButtonComponent id='delete' title='Delete'
onClick={onClickDelete}>Delete</ButtonComponent> <ScheduleComponent
ref={scheduleObj} width='100%' height='550px' selectedDate=
  {new Date(2018, 1, 15)} eventSettings={eventSettings}>
    <ViewsDirective>

```



```

        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
</ScheduleComponent>
</div>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import { useRef } from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
    ViewsDirective, ViewDirective
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { DataManager, Query } from '@syncfusion/ej2-data';
const App = () => {
    const scheduleObj = useRef<ScheduleComponent>(null);
    const scheduleData: Object[] = [{
        Id: 3,
        Subject: 'Testing',
        StartTime: new Date(2018, 1, 11, 9, 0),
        EndTime: new Date(2018, 1, 11, 10, 0),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=3'
    }, {
        Id: 4,
        Subject: 'Vacation',
        StartTime: new Date(2018, 1, 12, 11, 0),
        EndTime: new Date(2018, 1, 12, 12, 0),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
    }
    ];
    const eventSettings = { dataSource: scheduleData }
    const onClickAdd = (): void => {
        let Data: Object[] = [{
            Id: 1,
            Subject: 'Conference',
            StartTime: new Date(2018, 1, 15, 9, 0),
            EndTime: new Date(2018, 1, 15, 10, 0),
            IsAllDay: false,
            RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
        }
        ];
        scheduleObj.current.addEvent(Data);
    }
    const onClickSave = (): void => {

```

```

    let data: Object = new
DataManager(scheduleObj.current.getCurrentViewEvents()).executeLocal(new
Query().where('RecurrenceID', 'equal', 3));
    data[0].Subject = 'Occurrence edited';
    scheduleObj.current.saveEvent(data[0], 'EditOccurrence');
}
const onClickDelete = (): void => {
    let Data: Object[] = [{
        Id: 4,
        Subject: 'Vacation',
        RecurrenceID: 4,
        StartTime: new Date(2018, 1, 12, 11, 0),
        EndTime: new Date(2018, 1, 12, 12, 0),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
    }];
    scheduleObj.current.deleteEvent(Data, 'DeleteSeries');
}
return (
    <div>
        <ButtonComponent id='add' title='Add' onClick=
            {onClickAdd}>Add</ButtonComponent>
        <ButtonComponent id='edit' title='Edit'
onClick={onClickSave}>Edit</ButtonComponent>
        <ButtonComponent id='delete' title='Delete'
onClick={onClickDelete}>Delete</ButtonComponent> <ScheduleComponent
ref={scheduleObj} width='100%' height='550px' selectedDate=
    {new Date(2018, 1, 15)} eventSettings={eventSettings}>
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
    </div>
)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Syncfusion React Schedule</title>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0"
    />
        <meta name="description" content="Essential JS 2 for React
Components" />
        <meta name="author" content="Syncfusion" />
        <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>

<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>

</html>

```

When a single occurrence of the recurrence appointment is edited, `recurrenceID` field will be added which holds the `id` value of its parent recurrence appointment. It is applicable only for the edited occurrence appointments. Therefore the collection passing to the `saveEvent` with action as **EditOccurrence** should have `RecurrenceID` field as shown above.

Set default value for event fields in React Schedule component

Event window default fields name like Title, Location, etc.. can be customized and default value can be set to Subject field using `default` property which will be added if an appointment is created with empty subject.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
'@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const fieldsData = {
    subject: { title: 'Event Name', name: 'Subject', default: 'Add Name'
  },
    location: { title: 'Event Location', name: 'Location', default: 'USA'
  },
    description: { title: 'Summary', name: 'Description' },
    startTime: { title: 'From', name: 'StartTime' },
    endTime: { title: 'To', name: 'EndTime' }
  }
  const eventSettings = { dataSource: scheduleData, fields: fieldsData };
  return (<ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
  const fieldsData = {
    subject: { title: 'Event Name', name: 'Subject', default: 'Add Name' },
    location: { title: 'Event Location', name: 'Location', default: 'USA' },
    description: { title: 'Summary', name: 'Description' },
    startTime: { title: 'From', name: 'StartTime' },
    endTime: { title: 'To', name: 'EndTime' }
  }
  const eventSettings: EventSettingsModel = { dataSource: scheduleData,
fields: fieldsData };
  return (<ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
```

```

<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta name="description" content="Essential JS 2 for React Components" />
<meta name="author" content="Syncfusion" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Open event editor manually in React Schedule component

[Open Editor Window externally](#)

Schedule allows user to manually open the event editor on specific time or on certain events using `openEditor` method as shown below.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';

```

```

import { Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const eventButton = () => {
    let cellData = {
      startTime: new Date(2018, 1, 15, 10, 0),
      endTime: new Date(2018, 1, 15, 11, 0),
    };
    scheduleObj.current.openEditor(cellData, 'Add');
  }
  const eventEditorButton = () => {
    let eventData = {
      Id: 4,
      Subject: 'Meteor Showers in 2018',
      StartTime: new Date(2018, 1, 14, 13, 0),
      EndTime: new Date(2018, 1, 14, 14, 30)
    };
    scheduleObj.current.openEditor(eventData, 'Save');
  }
  return (<div>
    <ButtonComponent id='btn1' title='Click to open Editor'
onClick={eventButton}>Click to open Editor</ButtonComponent>
    <ButtonComponent id='btn2' title='Click to open Event Editor'
onClick={eventEditorButton}>Click to open Event Editor</ButtonComponent>
    <ScheduleComponent ref={scheduleObj} height='550px' selectedDate={new
Date(2018, 1, 15)}
      eventSettings={eventSettings} >
      <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
      </ViewsDirective>
      <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
  </div>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import {
  Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
ViewDirective, Inject, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {

```

```

const scheduleObj = useRef<ScheduleComponent>(null);
const eventSettings: EventSettingsModel = { dataSource: scheduleData };
const eventButton = (): void => {
  let cellData: Object = {
    startTime: new Date(2018, 1, 15, 10, 0),
    endTime: new Date(2018, 1, 15, 11, 0),
  };
  scheduleObj.current.openEditor(cellData, 'Add');
}
const eventEditorButton = (): void => {
  let eventData: Object = {
    Id: 4,
    Subject: 'Meteor Showers in 2018',
    StartTime: new Date(2018, 1, 14, 13, 0),
    EndTime: new Date(2018, 1, 14, 14, 30)
  };
  scheduleObj.current.openEditor(eventData, 'Save');
}
return (<div>
  <ButtonComponent id='btn1' title='Click to open Editor'
onClick={eventButton}>Click to open Editor</ButtonComponent>
  <ButtonComponent id='btn2' title='Click to open Event Editor'
onClick={eventEditorButton}>Click to open Event Editor</ButtonComponent>
  <ScheduleComponent ref={scheduleObj} height='550px' selectedDate={new
Date(2018, 1, 15)}
    eventSettings={eventSettings} >
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>
</div>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Open editor window on single click

By default, Scheduler Editor window will open when double clicking the cells or appointments. You can also open the editor window with single click by using `openEditor` method in `eventClick` and `cellClick` events of scheduler and setting `false` to `showQuickInfo`. The following example shows how to open editor window on single click of cells and appointments.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const scheduleObj = useRef(null);
    const eventSettings = { dataSource: scheduleData };
    const onCellClick = (args) => {
        scheduleObj.current.openEditor(args, 'Add');
    }
}

```



```

const onEventClick = (args) => {
  if (!args.event.RecurrenceRule) {
    scheduleObj.current.openEditor(args.event, 'Save');
  }
  else {
    scheduleObj.current.quickPopup.openRecurrenceAlert();
  }
}
return (<div>
  <ScheduleComponent ref={scheduleObj} height='550px' selectedDate={new
Date(2021, 7, 15)}
    eventSettings={eventSettings} showQuickInfo={false}
    eventClick={onEventClick}
    cellClick={onCellClick} >
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>
</div>)
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import {
  Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
  ViewDirective, CellClickEventArgs, EventSettingsModel, EventClickArgs, Inject
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from '../datasource';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const onCellClick = (args: CellClickEventArgs): void => {
    scheduleObj.current.openEditor(args, 'Add');
  }
  const onEventClick = (args: EventClickArgs): void => {
    if (!args.event as any).RecurrenceRule {
      scheduleObj.current.openEditor(args.event, 'Save');
    }
    else {
      scheduleObj.current.quickPopup.openRecurrenceAlert();
    }
  }
  return (<div>
    <ScheduleComponent ref={scheduleObj} height='550px' selectedDate={new
Date(2021, 7, 15)}
      eventSettings={eventSettings} showQuickInfo={false}
      eventClick={onEventClick}

```

```

        cellClick={onCellClick} >
        <ViewsDirective>
            <ViewDirective option='Day' />
            <ViewDirective option='Week' />
            <ViewDirective option='WorkWeek' />
            <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
</div>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;
            height: 40px;
            left: 45%;
            position: absolute;
            top: 45%;
            width: 30%;
        }
    </style>

```

```

    </style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Prevent date navigation in React Schedule component

We can prevent navigation while clicking on the date header by simply removing `e-navigate` class from header cells which can be achieved in the `renderCell` event as shown in the below demo.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { removeClass } from '@syncfusion/ej2-base';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onRenderCell = (args) => {
        if (args.elementType === "dateHeader" || args.elementType === "monthCells") {
            removeClass(args.element.childNodes, "e-navigate");
        }
    }
    return (
        <ScheduleComponent width='100%' height='500px' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings} currentView='WorkWeek' renderCell={onRenderCell}>
            <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
        </ScheduleComponent>
    );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    RenderCellEventArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { removeClass } from '@syncfusion/ej2-base';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    function onRenderCell(args: RenderCellEventArgs): void {
        if (args.elementType === "dateHeader" || args.elementType === "monthCells") {
            removeClass(args.element.childNodes, "e-navigate");
        }
    }

```

```

    }
  }
  return (<ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2018, 1, 15)} eventSettings={eventSettings} currentView='WorkWeek'
  renderCell={onRenderCell}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
  <script src="systemjs.config.js"></script>
  <style>
    #loader {
      color: #008cff;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>

```

```

        <div id='schedule'>
            <div id='loader'>Loading....</div>
        </div>
    </body>
</html>

```

Half yearly view in React Schedule component

The year view of our scheduler displays all the 365 days and their related appointments of a particular year. You can customize the year view by using the following properties.

- [firstMonthOfYear](#)
- [monthsCount](#)
- [monthHeaderTemplate](#)

In the following code example, you can see how to render only the last six months of a year in the scheduler. To start with the month of June, `firstMonthYear` is set to 6 and `monthsCount` is set to 6 to render only 6 months.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, ViewsDirective, ViewDirective, Resize,
DragAndDrop, ResourcesDirective, ResourceDirective, Inject, Year as YearView,
TimelineYear } from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: resourceData };
    const group = { resources: ['Categories'] };
    const categoriesData = [
        { text: 'Nancy', id: 1, color: '#ffaa00' },
        { text: 'Steven', id: 2, color: '#f8a398' },
        { text: 'Robert', id: 3, color: '#7499e1' },
        { text: 'Smith', id: 4, color: '#5978ee' },
        { text: 'Micheal', id: 5, color: '#df5286' }
    ];
    const getMonthHeaderText = (props) => {
        return (<div>{props.date.toLocaleString('en-us', { month: 'long' })} + ' '
+ props.date.getFullYear())</div>);
    }
    const resourceHeaderTemplate = (props) => {
        return (<div className="template-wrap">
            <div className="resource-details">
                <div className="resource-name">{props.resourceData.text}</div>
            </div>
        </div>);
    }
    return (<ScheduleComponent width="100%" height="495px" selectedDate={new
Date(2021, 7, 15)} eventSettings={eventSettings} firstMonthOfYear={6}
monthsCount={6} group={group} resourceHeaderTemplate={resourceHeaderTemplate}
monthHeaderTemplate={getMonthHeaderText}>
        <ResourcesDirective>

```

```

    <ResourceDirective field="TaskId" title="Category" name="Categories"
allowMultiple={true} dataSource={categoriesData} textField="text"
idField="id" colorField="color" />
  </ResourcesDirective>
  <ViewsDirective>
    <ViewDirective option="Year" />
    <ViewDirective option="TimelineYear" displayName="Horizontal
TimelineYear" isSelected={true} />
    <ViewDirective option="TimelineYear" displayName="Vertical
TimelineYear" orientation="Vertical" />
  </ViewsDirective>
  <Inject services={[YearView, TimelineYear, Resize, DragAndDrop]} />
</ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, ViewsDirective, ViewDirective, Resize, DragAndDrop,
  ResourcesDirective, ResourceDirective, Inject, Year as YearView, TimelineYear
} from '@syncfusion/ej2-react-schedule';
import { resourceData } from './datasource';
const App = () => {
  const eventSettings = { dataSource: resourceData };
  const group = { resources: ['Categories'] };
  const categoriesData: Object[] = [
    { text: 'Nancy', id: 1, color: '#ffaa00' },
    { text: 'Steven', id: 2, color: '#f8a398' },
    { text: 'Robert', id: 3, color: '#7499e1' },
    { text: 'Smith', id: 4, color: '#5978ee' },
    { text: 'Micheal', id: 5, color: '#df5286' }
  ];
  const getMonthHeaderText = (props): JSX.Element => {
    return (<div>{props.date.toLocaleString('en-us', { month: 'long' })} + ' '
+ props.date.getFullYear()</div>);
  }
  const resourceHeaderTemplate = (props): JSX.Element => {
    return (
      <div className="template-wrap">
        <div className="resource-details">
          <div className="resource-name">{props.resourceData.text}</div>
        </div>
      </div>
    );
  }
  return (<ScheduleComponent
    width="100%"
    height="495px"
    selectedDate={new Date(2021, 7, 15)}
    eventSettings={eventSettings}
    firstMonthOfYear={6}

```

```

    monthsCount={6}
    group={group}
    resourceHeaderTemplate={resourceHeaderTemplate}
    monthHeaderTemplate={getMonthHeaderText}
  >
  <ResourcesDirective>
    <ResourceDirective
      field="TaskId"
      title="Category"
      name="Categories"
      allowMultiple={true}
      dataSource={categoriesData}
      textField="text"
      idField="id"
      colorField="color"
    />
  </ResourcesDirective>
  <ViewsDirective>
    <ViewDirective option="Year" />
    <ViewDirective
      option="TimelineYear"
      displayName="Horizontal TimelineYear"
      isSelected={true}
    />
    <ViewDirective
      option="TimelineYear"
      displayName="Vertical TimelineYear"
      orientation="Vertical"
    />
  </ViewsDirective>
  <Inject
    services={[YearView, TimelineYear, Resize, DragAndDrop]}
  />
</ScheduleComponent>
);
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-schedule .e-vertical-view .e-resource-cells {
        height: 62px;
    }
    .e-schedule .template-wrap {
        display: flex;
        text-align: left;
    }
    .e-schedule .template-wrap .resource-details {
        padding-left: 10px;
    }
    .e-schedule .template-wrap .resource-details .resource-name {
        font-size: 18px;
        font-weight: 500;
        margin-top: 5px;
    }
    .e-schedule.e-device .template-wrap .resource-details .resource-name
{
        font-size: inherit;
        font-weight: inherit;
    }
    .e-schedule.e-device .e-resource-tree-popup .e-fullrow {
        height: 50px;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```


Set different event time duration in React Schedule component

In event window, start/end time duration will be processed based on the **interval** value within the **timeScale** property. By default, **interval** value is 30, therefore in event window start/end time duration will be in 30 mins duration. You can set custom interval range to the start/end time in event window using **popupOpen** event as shown below.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings = { dataSource: scheduleData };
    const onPopupOpen = (args) => {
        args.duration = 40;
    }
    return <ScheduleComponent width='100%' height='500px' selectedDate={new Date(2018, 1, 15)} eventSettings={eventSettings} popupOpen={onPopupOpen}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>;
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
    PopupOpenEventArgs, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
const App = () => {
    const eventSettings: EventSettingsModel = { dataSource: scheduleData };
    const onPopupOpen = (args: PopupOpenEventArgs): void => {
        args.duration = 40;
    }
    return <ScheduleComponent width='100%' height='500px' selectedDate=
        {new Date(2018, 1, 15)} eventSettings={eventSettings}
        popupOpen={onPopupOpen}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
    </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```

<title>Syncfusion React Schedule</title>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta name="description" content="Essential JS 2 for React Components" />
<meta name="author" content="Syncfusion" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Set different work hours in React Schedule component

By default, the work hours of the Scheduler is highlighted based on the start and end values provided within the `workHours` property which remains same for all days. To highlight different work hours range for different days, `setWorkHours` method can be used as follows.

INDEX.JSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';

```

```

import { Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
ViewDirective, Inject } from '@syncfusion/ej2-react-schedule';
import { scheduleData } from './datasource';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: scheduleData };
  const onClick = () => {
    if (scheduleObj.current) {
      let dates = [new Date(2018, 1, 15), new Date(2018, 1, 17)];
      scheduleObj.current.setWorkHours(dates, '11:00', '20:00');
    }
  }
  return (
    <div>
      <ButtonComponent title='Set work hours' onClick={onClick}>Click to set
work hours</ButtonComponent>
      <ScheduleComponent ref={scheduleObj} height='550px' selectedDate={new
Date(2018, 1, 15)}
        workHours={{ highlight: true, start: '09:00', end: '11:00' }}
        eventSettings={eventSettings}>
        <ViewsDirective>
          <ViewDirective option='Day' />
          <ViewDirective option='Week' />
          <ViewDirective option='WorkWeek' />
          <ViewDirective option='Month' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month]} />
      </ScheduleComponent>
    </div>
  );
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import * as ReactDOM from 'react-dom';
import * as React from 'react';
import { useRef } from 'react';
import { Day, Week, WorkWeek, Month, ScheduleComponent, ViewsDirective,
EventSettingsModel, ViewDirective, Inject } from '@syncfusion/ej2-react-
schedule';
import { scheduleData } from './datasource';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const onClick = (): void => {
    if (scheduleObj) {
      let dates: Date[] = [new Date(2018, 1, 15), new Date(2018, 1, 17)];
      scheduleObj.current.setWorkHours(dates, '11:00', '20:00');
    }
  }
  return (
    <div>

```

```

    <ButtonComponent title='Set work hours' onClick={onClick}>Click to set
work hours</ButtonComponent>
    <ScheduleComponent ref={scheduleObj} height='550px' selectedDate={new
Date(2018, 1, 15) }
        workHours={{ highlight: true, start: '09:00', end: '11:00' }}
        eventSettings={eventSettings}>
    <ViewsDirective>
        <ViewDirective option='Day' />
        <ViewDirective option='Week' />
        <ViewDirective option='WorkWeek' />
        <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
    </ScheduleComponent>
</div>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Essential JS 2 for React Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
    <script src="systemjs.config.js"></script>
    <style>
        #loader {
            color: #008cff;

```

```

        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Show quick info template in React Schedule component

This demo showcases the quick popups for cells and appointments with the customized templates.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, ResourcesDirective, ResourceDirective, Day, Week,
WorkWeek, Month, Agenda, MonthAgenda, Inject } from '@syncfusion/ej2-react-
schedule';
import { Internationalization } from '@syncfusion/ej2-base';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { TextBoxComponent } from '@syncfusion/ej2-react-inputs';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { ScheduleData } from '../datasource';
const App = () => {
    const eventTypeObj = useRef(null);
    const titleObj = useRef(null);
    const notesObj = useRef(null);
    const scheduleObj = useRef(null);
    const intl = new Internationalization();
    const roomData = [
        { Name: 'Jammy', Id: 1, Capacity: 20, Color: '#ea7a57', Type:
'Conference' },
        { Name: 'Tweety', Id: 2, Capacity: 7, Color: '#7fa900', Type: 'Cabin' },
        { Name: 'Nestle', Id: 3, Capacity: 5, Color: '#5978ee', Type: 'Cabin' },
        { Name: 'Phoenix', Id: 4, Capacity: 15, Color: '#fec200', Type:
'Conference' },
        { Name: 'Mission', Id: 5, Capacity: 25, Color: '#df5286', Type:
'Conference' },
        { Name: 'Hangout', Id: 6, Capacity: 10, Color: '#00bdae', Type: 'Cabin'
},
        { Name: 'Rick Roll', Id: 7, Capacity: 20, Color: '#865fcf', Type:
'Conference' },
        { Name: 'Rainbow', Id: 8, Capacity: 8, Color: '#1aaa55', Type: 'Cabin' },
        { Name: 'Swarm', Id: 9, Capacity: 30, Color: '#df5286', Type:
'Conference' },
        { Name: 'Photogenic', Id: 10, Capacity: 25, Color: '#710193', Type:
'Conference' }
    ];

```

```

const eventSettings = { dataSource: ScheduleData };
const headerTemplate = (props) => {
    return (<div className="quick-info-header">
        <div className="quick-info-header-content"
style={getHeaderStyles(props)}>
            <div className="quick-info-title">{getHeaderTitle(props)}</div>
            <div className="duration-text">{getHeaderDetails(props)}</div>
        </div>
    </div>);
}
const contentTemplate = (props) => {
    return (<div className="quick-info-content">
        {props.elementType === 'cell' ?
            <div className="e-cell-content">
                <div className="content-area">
                    <TextBoxComponent id="title" ref={titleObj} placeholder="Title"
/>
                </div>
                <div className="content-area">
                    <DropDownListComponent id="eventType" ref={eventTypeObj}
dataSource={roomData} fields={{ text: "Name", value: "Id" }}
placeholder="Choose Type" index={0} popupHeight="200px" />
                </div>
                <div className="content-area">
                    <TextBoxComponent id="notes" ref={notesObj} placeholder="Notes"
/>
                </div>
            </div>
        :
        <div className="event-content">
            <div className="meeting-type-wrap">
                <label>Subject</label>:
                <span>{props.Subject}</span>
            </div>
            <div className="meeting-subject-wrap">
                <label>Type</label>:
                <span>{getEventType(props)}</span>
            </div>
            <div className="notes-wrap">
                <label>Notes</label>:
                <span>{props.Description}</span>
            </div>
        </div>
    </div>);
}
const footerTemplate = (props) => {
    return (<div className="quick-info-footer">
        {props.elementType === "cell" ?
            <div className="cell-footer">
                <ButtonComponent id="more-details" cssClass='e-flat' content="More
Details" onClick={buttonClickActions.bind(this)} />
                <ButtonComponent id="add" cssClass='e-flat' content="Add"
isPrimary={true} onClick={buttonClickActions.bind(this)} />
            </div>
        :
        <div className="event-footer">

```

```

        <ButtonComponent id="delete" cssClass='e-flat' content="Delete"
onClick={buttonClickActions.bind(this)} />
        <ButtonComponent id="more-details" cssClass='e-flat' content="More
Details" isPrimary={true} onClick={buttonClickActions.bind(this).bind(this)}
/>
    </div>}
</div>);
}
const quickInfoTemplates = {
    header: headerTemplate,
    content: contentTemplate,
    footer: footerTemplate
};
const getResourceData = (data) => {
    const resources = scheduleObj.current.getResourceCollections().slice(-
1)[0];
    const resourceData = resources.dataSource.filter((resource) =>
resource.Id === data.RoomId)[0];
    return resourceData;
}
const getHeaderStyles = (data) => {
    if (data.elementType === 'cell') {
        return { alignItems: 'center', color: '#919191' };
    }
    else {
        const resourceData = getResourceData(data);
        return { background: resourceData.Color, color: '#FFFFFF' };
    }
}
const getHeaderTitle = (data) => {
    return (data.elementType === 'cell') ? 'Add Appointment' : 'Appointment
Details';
}
const getHeaderDetails = (data) => {
    return intl.formatDate(data.StartTime, { type: 'date', skeleton: 'full'
}) + ' (' +
        intl.formatDate(data.StartTime, { skeleton: 'hm' }) + ' - ' +
        intl.formatDate(data.EndTime, { skeleton: 'hm' }) + ')';
}
const getEventType = (data) => {
    const resourceData = getResourceData(data);
    return resourceData.Name;
}
const buttonClickActions = (e) => {
    const quickPopup = scheduleObj.current.element.querySelector('.e-quick-
popup-wrapper');
    const getSlotData = () => {
        const cellDetails =
scheduleObj.current.getCellDetails(scheduleObj.current.getSelectedElements())
;
        const addObj = {};
        addObj.Id = scheduleObj.current.getEventMaxID();
        addObj.Subject = titleObj.current.value;
        addObj.StartTime = new Date(+cellDetails.startTime);
        addObj.EndTime = new Date(+cellDetails.endTime);
        addObj.Description = notesObj.current.value;
        addObj.RoomId = eventTypeObj.current.value;
    }

```

```

    return addObj;
  };
  if (e.target.id === 'add') {
    const addObj = getSlotData();
    scheduleObj.current.addEvent(addObj);
  }
  else if (e.target.id === 'delete') {
    const eventDetails = scheduleObj.current.activeEventData.event;
    let currentAction = 'Delete';
    if (eventDetails.RecurrenceRule) {
      currentAction = 'DeleteOccurrence';
    }
    scheduleObj.current.deleteEvent(eventDetails, currentAction);
  }
  else {
    const isCellPopup = quickPopup.firstElementChild.classList.contains('e-cell-popup');
    const eventDetails = isCellPopup ? getSlotData() :
      scheduleObj.current.activeEventData.event;
    let currentAction = isCellPopup ? 'Add' : 'Save';
    if (eventDetails.RecurrenceRule) {
      currentAction = 'EditOccurrence';
    }
    scheduleObj.current.openEditor(eventDetails, currentAction, true);
  }
  scheduleObj.current.closeQuickInfoPopup();
}
return (<div className='schedule-control-section'>
  <div className='col-lg-12 control-section'>
    <div className='control-wrapper'>
      <ScheduleComponent id="schedule" cssClass='quick-info-template'
ref={scheduleObj} height="650px" selectedDate={new Date(2020, 0, 9)}
eventSettings={eventSettings} quickInfoTemplates={quickInfoTemplates}>
        <ResourcesDirective>
          <ResourceDirective field='RoomId' title='Room Type'
name='MeetingRoom' textField='Name' idField='Id' colorField='Color'
dataSource={roomData}></ResourceDirective>
        </ResourcesDirective>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda,
MonthAgenda]} />
      </ScheduleComponent>
    </div>
  </div>
</div>);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {

```



```

ScheduleComponent, ResourcesDirective, ResourceDirective, Day, Week,
WorkWeek, Month,
Agenda, MonthAgenda, Inject, ResourcesModel, CellClickEventArgs,
CurrentAction, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { Internationalization } from '@syncfusion/ej2-base';
import { DropDownListComponent } from '@syncfusion/ej2-react-dropdowns';
import { TextBoxComponent } from '@syncfusion/ej2-react-inputs';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { ScheduleData } from '../datasource';
const App = () => {
  const eventTypeObj = useRef<DropDownListComponent>(null);
  const titleObj = useRef<TextBoxComponent>(null);
  const notesObj = useRef<TextBoxComponent>(null);
  const scheduleObj = useRef<ScheduleComponent>(null);
  const intl: Internationalization = new Internationalization();
  const headerTemplate = (props: { [key: string]: Date }): JSX.Element => {
    return (
      <div className="quick-info-header">
        <div className="quick-info-header-content"
style={getHeaderStyles(props)}>
          <div className="quick-info-title">{getHeaderTitle(props)}</div>
          <div className="duration-text">{getHeaderDetails(props)}</div>
        </div>
      </div>
    );
  }
  const contentTemplate = (props: { [key: string]: string }): JSX.Element => {
    return (
      <div className="quick-info-content">
        {props.elementType === 'cell' ?
          <div className="e-cell-content">
            <div className="content-area">
              <TextBoxComponent id="title" ref={titleObj} placeholder="Title"
/>
            </div>
            <div className="content-area">
              <DropDownListComponent id="eventType" ref={eventTypeObj}
dataSource={roomData}
fields={{ text: "Name", value: "Id" }} placeholder="Choose
Type" index={0} popupHeight="200px" />
            </div>
            <div className="content-area">
              <TextBoxComponent id="notes" ref={notesObj} placeholder="Notes"
/>
            </div>
          </div>
          :
          <div className="event-content">
            <div className="meeting-type-wrap">
              <label>Subject</label>:
              <span>{props.Subject}</span>
            </div>
            <div className="meeting-subject-wrap">
              <label>Type</label>:
              <span>{getEventType(props)}</span>
            </div>
          </div>
        }
      </div>
    );
  }

```

```

        </div>
        <div className="notes-wrap">
            <label>Notes</label>:
            <span>{props.Description}</span>
        </div>
    </div>
    }
</div>
);
}
const footerTemplate = (props: { [key: string]: Object }): JSX.Element => {
    return (
        <div className="quick-info-footer">
            {props.elementType === "cell" ?
                <div className="cell-footer">
                    <ButtonComponent id="more-details" cssClass='e-flat'
content="More Details" onClick={buttonClickActions.bind(this)} />
                    <ButtonComponent id="add" cssClass='e-flat' content="Add"
isPrimary={true} onClick={buttonClickActions.bind(this)} />
                </div>
                :
                <div className="event-footer">
                    <ButtonComponent id="delete" cssClass='e-flat' content="Delete"
onClick={buttonClickActions.bind(this)} />
                    <ButtonComponent id="more-details" cssClass='e-flat'
content="More Details" isPrimary={true}
onClick={buttonClickActions.bind(this).bind(this)} />
                </div>
            }
        </div>
    );
}
const roomData: { [key: string]: Object }[] = [
    { Name: 'Jammy', Id: 1, Capacity: 20, Color: '#ea7a57', Type:
'Conference' },
    { Name: 'Tweety', Id: 2, Capacity: 7, Color: '#7fa900', Type: 'Cabin' },
    { Name: 'Nestle', Id: 3, Capacity: 5, Color: '#5978ee', Type: 'Cabin' },
    { Name: 'Phoenix', Id: 4, Capacity: 15, Color: '#fec200', Type:
'Conference' },
    { Name: 'Mission', Id: 5, Capacity: 25, Color: '#df5286', Type:
'Conference' },
    { Name: 'Hangout', Id: 6, Capacity: 10, Color: '#00bdae', Type: 'Cabin'
},
    { Name: 'Rick Roll', Id: 7, Capacity: 20, Color: '#865fcf', Type:
'Conference' },
    { Name: 'Rainbow', Id: 8, Capacity: 8, Color: '#1aaa55', Type: 'Cabin' },
    { Name: 'Swarm', Id: 9, Capacity: 30, Color: '#df5286', Type:
'Conference' },
    { Name: 'Photogenic', Id: 10, Capacity: 25, Color: '#710193', Type:
'Conference' }
];
const eventSettings: EventSettingsModel = { dataSource: ScheduleData };
const quickInfoTemplates = {
    header: headerTemplate,
    content: contentTemplate,
    footer: footerTemplate
};

```

```

const getResourceData = (data: { [key: string]: Object }): { [key: string]:
Object } => {
    const resources: ResourcesModel =
scheduleObj.current.getResourceCollections().slice(-1)[0];
    const resourceData: { [key: string]: Object } = (resources.dataSource as
Object[]).filter((resource: { [key: string]: Object }) =>
    resource.Id === data.RoomId)[0] as { [key: string]: Object };
    return resourceData;
}
const getHeaderStyles = (data: { [key: string]: Object }): Object => {
    if (data.elementType === 'cell') {
        return { alignItems: 'center', color: '#919191' };
    } else {
        const resourceData: { [key: string]: Object } = getResourceData(data);
        return { background: resourceData.Color, color: 'FFFFFF' };
    }
}
const getHeaderTitle = (data: { [key: string]: Object }): string => {
    return (data.elementType === 'cell') ? 'Add Appointment' : 'Appointment
Details';
}
const getHeaderDetails = (data: { [key: string]: Date }): string => {
    return intl.formatDate(data.StartTime, { type: 'date', skeleton: 'full'
}) + ' (' +
        intl.formatDate(data.StartTime, { skeleton: 'hm' }) + ' - ' +
        intl.formatDate(data.EndTime, { skeleton: 'hm' }) + ')';
}
const getEventType = (data: { [key: string]: string }): string => {
    const resourceData: { [key: string]: Object } = getResourceData(data);
    return resourceData.Name as string;
}
const buttonClickActions = (e: Event) => {
    const quickPopup: HTMLElement =
scheduleObj.current.element.querySelector('.e-quick-popup-wrapper') as
HTMLElement;
    const getSlotData: Function = (): { [key: string]: Object } => {
        const cellDetails: CellClickEventArgs =
scheduleObj.current.getCellDetails(scheduleObj.current.getSelectedElements())
;
        const addObj: { [key: string]: Object } = {};
        addObj.Id = scheduleObj.current.getEventMaxID();
        addObj.Subject = titleObj.current.value;
        addObj.StartTime = new Date(+cellDetails.startTime);
        addObj.EndTime = new Date(+cellDetails.endTime);
        addObj.Description = notesObj.current.value;
        addObj.RoomId = eventTypeObj.current.value;
        return addObj;
    };
    if ((e.target as HTMLElement).id === 'add') {
        const addObj: { [key: string]: Object } = getSlotData();
        scheduleObj.current.addEvent(addObj);
    } else if ((e.target as HTMLElement).id === 'delete') {
        const eventDetails: { [key: string]: Object } =
scheduleObj.current.activeEventData.event as { [key: string]: Object };
        let currentAction: CurrentAction = 'Delete';
        if (eventDetails.RecurrenceRule) {
            currentAction = 'DeleteOccurrence';
        }
    }
}

```

```

    }
    scheduleObj.current.deleteEvent(eventDetails, currentAction);
  } else {
    const isCellPopup: boolean = (quickPopup.firstElementChild as
HTMLInputElement).classList.contains('e-cell-popup');
    const eventDetails: { [key: string]: Object } = isCellPopup ?
getSlotData() :
    scheduleObj.current.activeEventData.event as { [key: string]: Object
};
    let currentAction: CurrentAction = isCellPopup ? 'Add' : 'Save';
    if (eventDetails.RecurrenceRule) {
      currentAction = 'EditOccurrence';
    }
    scheduleObj.current.openEditor(eventDetails, currentAction, true);
  }
  scheduleObj.current.closeQuickInfoPopup();
}
return (
  <div className='schedule-control-section'>
    <div className='col-lg-12 control-section'>
      <div className='control-wrapper'>
        <ScheduleComponent id="schedule" cssClass='quick-info-template'
ref={scheduleObj} height="650px"
        selectedDate={new Date(2020, 0, 9)} eventSettings={eventSettings}
quickInfoTemplates={quickInfoTemplates}>
          <ResourcesDirective>
            <ResourceDirective field='RoomId' title='Room Type'
name='MeetingRoom' textField='Name' idField='Id'
            colorField='Color' dataSource={roomData}></ResourceDirective>
          </ResourcesDirective>
          <Inject services={[Day, Week, WorkWeek, Month, Agenda,
MonthAgenda]} />
        </ScheduleComponent>
      </div>
    </div>
  </div>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
  />
    <meta name="description" content="Essential JS 2 for React
Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>

<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>

<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>

</html>

```

Enable scroll option on all day section in React Schedule component

When you have larger number of appointments in all-day row, it is difficult to view all the appointments properly. In that case you can enable scroller option for all-day row by setting true to **enableAllDayScroll** whereas its default value is false. When setting this property to true, individual scroller for all-day row is enabled when it reaches its maximum height on expanding.

Note: This property is not applicable for Scheduler with height **auto**.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from
 '@syncfusion/ej2-react-schedule';

```

```
import { generateObject } from './datasource';
const App = () => {
  const data = generateObject();
  const eventSettings = { dataSource: data };
  return <ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2021, 3, 28)} enableAllDayScroll={true} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>;
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject,
  EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { generateObject } from './datasource';
const App = () => {
  const data: Object[] = generateObject() as Object[];
  const eventSettings: EventSettingsModel = { dataSource: data };
  return <ScheduleComponent width='100%' height='500px' selectedDate={new
Date(2021, 3, 28)} enableAllDayScroll={true} eventSettings={eventSettings}>
    <Inject services={[Day, Week, WorkWeek, Month, Agenda]} />
  </ScheduleComponent>
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Schedule</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Essential JS 2 for React Components" />
  <meta name="author" content="Syncfusion" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>
<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Manual refresh in React Schedule component

Refresh Template

In Scheduler, we can able to refresh the elements of the template alone instead of the entire scheduler by using the [refreshTemplates](#) public method. We can provide an additional option to refresh specific templates alone or all templates together by using this method. The following template names are accepted as an argument to refresh it specifically.

- eventTemplate
- dateHeaderTemplate
- resourceHeaderTemplate
- quickInfoTemplates
- editorTemplate
- tooltipTemplate
- headerTooltipTemplate

In the following code example, you can see how to use the refreshTemplates method to refresh multiple templates. Here, we have added the following scheduler templates such as [cellTemplate](#), [dateHeaderTemplate](#), [eventTemplate](#) and [resourceHeaderTemplate](#)

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```

import { ScheduleComponent, ViewsDirective, ViewDirective, Day, Week,
WorkWeek, Month, TimelineMonth, Inject, Resize, DragAndDrop,
ResourcesDirective, ResourceDirective } from '@syncfusion/ej2-react-
schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { Internationalization } from '@syncfusion/ej2-base';
import { webinarData } from './datasource';
const App = () => {
  const scheduleObj = useRef(null);
  const eventSettings = { dataSource: webinarData };
  const group = { resources: ['Doctors'] }
  const instance = new Internationalization();
  const resourceData = [
    { text: 'Will Smith', id: 1, color: '#ea7a57', workDays: [1, 2, 4, 5],
startHour: '08:00', endHour: '15:00' },
    { text: 'Alice', id: 2, color: 'rgb(53, 124, 210)', workDays: [1, 3, 5],
startHour: '08:00', endHour: '17:00' },
    { text: 'Robson', id: 3, color: '#7fa900', startHour: '08:00', endHour:
'16:00' }
  ];
  const getTimeString = (value) => {
    return instance.formatDate(value, { skeleton: 'hm' });
  }
  const eventTemplate = (props) => {
    return (<div className="app-template-wrap" style={{ background:
props.SecondaryColor }}>
      <div className="subject" style={{ background: props.PrimaryColor
}}>{props.Subject}</div>
      <div className="time" style={{ background: props.PrimaryColor }}>
        Time: {getTimeString(props.StartTime)} -
{getTimeString(props.EndTime)}</div>
      <div className="image"><img
src={"https://ej2.syncfusion.com/demos/src/schedule/images/" +
props.ImageName + ".svg"} alt={props.ImageName} /></div>
      <div className="event-description">{props.Description}</div>
      <div className="footer" style={{ background: props.PrimaryColor
}}></div></div></div>);
  }
  const getDoctorImage = (value) => {
    return getDoctorName(value).replace(' ', '-').toLowerCase();
  }
  const getDoctorName = (value) => {
    return ((value.resourceData) ?
      value.resourceData[value.resource.textField] :
      value.resourceName);
  }
  const getDoctorLevel = (value) => {
    let resourceName = getDoctorName(value);
    return (resourceName === 'Will Smith') ? 'Cardiologist' : (resourceName
=== 'Alice') ? 'Neurologist' : 'Orthopedic Surgeon';
  }
  const getDateHeaderText = (value) => {
    return instance.formatDate(value, { skeleton: 'Ed' });
  }
  const getWeather = (value) => {
    switch (value.getDay()) {
      case 0:

```



```

        return '<img class="weather-image" src=
"https://ej2.syncfusion.com/demos/src/schedule/images/weather-clear.svg"
/><div class="weather-text">25°C</div>';
    case 1:
        return '<div class="weather-text">18°C</div>';
    case 2:
        return '<div class="weather-text">10°C</div>';
    case 3:
        return '<div class="weather-text">16°C</div>';
    case 4:
        return '<div class="weather-text">8°C</div>';
    case 5:
        return '<div class="weather-text">27°C</div>';
    case 6:
        return '<div class="weather-text">17°C</div>';
    default:
        return null;
    }
}

const dateHeaderTemplate = (props) => {
    return (<div><div>{getDateHeaderText(props.date)}</div><div
className="date-text" dangerouslySetInnerHTML={{ __html:
getWeather(props.date) }}></div></div>);
}

const resourceHeaderTemplate = (props) => {
    return (<div className="res-template-wrap"><div className={"resource-
image " + getDoctorImage(props)}></div>
    <div className="resource-detail"><div className="resource-
name">{getDoctorName(props)}</div>
    <div className="resource-
designation">{getDoctorLevel(props)}</div></div></div>);
}

const getMonthCellText = (date) => {
    if (date.getMonth() === 1 && date.getDate() === 23) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    }
    else if (date.getMonth() === 1 && date.getDate() === 9) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/get-together.svg" />';
    }
    else if (date.getMonth() === 1 && date.getDate() === 13) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    }
}

```

```

    else if (date.getMonth() === 1 && date.getDate() === 22) {
      return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/thanksgiving-day.svg"
/>';
    }
    else if (date.getMonth() === 1 && date.getDate() === 14) {
      return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    }
    return '';
  }
  const getWorkCellText = (date) => {
    let weekEnds = [0, 6];
    if (weekEnds.indexOf(date.getDay()) >= 0) {
      return "<span class='caption'>Weekend</span>";
    }
    return '';
  }
  const cellTemplate = (props) => {
    if (props.type === "workCells") {
      return (<div className="cell-template-wrap" dangerouslySetInnerHTML={{
        __html: getWorkCellText(props.date) }}></div>);
    }
    if (props.type === "monthCells") {
      return (<div className="cell-template-wrap" dangerouslySetInnerHTML={{
        __html: getMonthCellText(props.date) }}></div>);
    }
    return (<div></div>);
  }
  const refreshCellTemplate = () => {
    scheduleObj.current.refreshTemplates("cellTemplate");
  }
  const refreshDateHeaderTemplate = () => {
    scheduleObj.current.refreshTemplates("dateHeaderTemplate");
  }
  const refreshEventTemplate = () => {
    scheduleObj.current.refreshTemplates("eventTemplate");
  }
  const refreshResHeaderTemplate = () => {
    scheduleObj.current.refreshTemplates("resourceHeaderTemplate");
  }
  const refreshAllTemplate = () => {
    scheduleObj.current.refreshTemplates();
  }
  return (<div className='schedule-control-section'>
    <div className='control-section'>
      <div className='control-wrapper'>
        <div style={{ "display": 'flex' }}>
          <div style={{ paddingRight: '10px' }}>
            <ButtonComponent cssClass='e-info'
onClick={refreshCellTemplate}>Refresh cellTemplate</ButtonComponent>
          </div>
          <div style={{ paddingRight: '10px' }}>
            <ButtonComponent cssClass='e-info'
onClick={refreshDateHeaderTemplate}>Refresh
dateHeaderTemplate</ButtonComponent>
          </div>
        </div>
      </div>
    </div>
  )

```

```

        <div style={{ paddingRight: '10px' }}>
            <ButtonComponent cssClass='e-info'
onClick={refreshEventTemplate}>Refresh eventTemplate</ButtonComponent>
        </div>
        <div style={{ paddingRight: '10px' }}>
            <ButtonComponent cssClass='e-info'
onClick={refreshResHeaderTemplate}>Refresh
resourceHeaderTemplate</ButtonComponent>
        </div>
        <div style={{ paddingRight: '10px' }}>
            <ButtonComponent cssClass='e-info'
onClick={refreshAllTemplate}>Refresh All Templates</ButtonComponent>
        </div>
    </div>
    <ScheduleComponent width='100%' height='650px' cssClass='schedule-
date-header-template' ref={scheduleObj} selectedDate={new Date(2021, 1, 15)}
readonly={true} eventSettings={eventSettings}
dateHeaderTemplate={dateHeaderTemplate}
resourceHeaderTemplate={resourceHeaderTemplate} cellTemplate={cellTemplate}
group={group}>
        <ResourcesDirective>
            <ResourceDirective field='DoctorId' title='Doctor Name'
name='Doctors' dataSource={resourceData} textField='text' idField='id'
groupIDField='groupId' colorField='color' workDaysField='workDays'
startHourField='startHour' endHourField='endHour'>
                </ResourceDirective>
            </ResourcesDirective>
        <ViewsDirective>
            <ViewDirective option='Week' eventTemplate={eventTemplate} />
            <ViewDirective option='Month' />
            <ViewDirective option='TimelineMonth' />
        </ViewsDirective>
        <Inject services={[Day, Week, WorkWeek, Month, TimelineMonth,
Resize, DragAndDrop]} />
    </ScheduleComponent>
</div>
</div>
</div>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
    ScheduleComponent, ViewsDirective, ViewDirective, Day, Week, WorkWeek,
    Month, TimelineMonth,
    Inject, Resize, DragAndDrop, ResourcesDirective, GroupModel,
    ResourceDirective, ResourceDetails, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { Internationalization } from '@syncfusion/ej2-base';

```

```

import { webinarData } from './datasource';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const eventSettings: EventSettingsModel = { dataSource: webinarData };
  const group: GroupModel = { resources: ['Doctors'] }
  const instance: Internationalization = new Internationalization();
  const resourceData: Record<string, any>[] = [
    { text: 'Will Smith', id: 1, color: '#ea7a57', workDays: [1, 2, 4, 5],
startHour: '08:00', endHour: '15:00' },
    { text: 'Alice', id: 2, color: 'rgb(53, 124, 210)', workDays: [1, 3, 5],
startHour: '08:00', endHour: '17:00' },
    { text: 'Robson', id: 3, color: '#7fa900', startHour: '08:00', endHour:
'16:00' }
  ];
  const getTimeString = (value: Date) => {
    return instance.formatDate(value, { skeleton: 'hm' });
  }
  const eventTemplate = (props): JSX.Element => {
    return (<div className="app-template-wrap" style={{ background:
props.SecondaryColor }}>
      <div className="subject" style={{ background: props.PrimaryColor
}}>{props.Subject}</div>
      <div className="time" style={{ background: props.PrimaryColor }}>
        Time: {getTimeString(props.StartTime)} -
{getTimeString(props.EndTime)}</div>
      <div className="image"><img
src={{"https://ej2.syncfusion.com/demos/src/schedule/images/" +
props.ImageName + ".svg"}} alt={props.ImageName} /></div>
      <div className="event-description">{props.Description}</div>
      <div className="footer" style={{ background: props.PrimaryColor
}}></div></div>
    );
  }
  const getDoctorImage = (value: ResourceDetails): string => {
    return getDoctorName(value).replace(' ', '-').toLowerCase();
  }
  const getDoctorName = (value: ResourceDetails): string => {
    return (((value as ResourceDetails).resourceData) ?
(value as ResourceDetails).resourceData[(value as
ResourceDetails).resource.textField] :
value.resourceName) as string;
  }
  const getDoctorLevel = (value: ResourceDetails): string => {
    let resourceName: string = getDoctorName(value);
    return (resourceName === 'Will Smith') ? 'Cardiologist' : (resourceName
=== 'Alice') ? 'Neurologist' : 'Orthopedic Surgeon';
  }
  const getDateHeaderText = (value: Date): string => {
    return instance.formatDate(value, { skeleton: 'Ed' });
  }
  const getWeather = (value: Date) => {
    switch (value.getDay()) {
      case 0:
        return '<img class="weather-image" src=
"https://ej2.syncfusion.com/demos/src/schedule/images/weather-clear.svg"
/><div class="weather-text">25°C</div>';
      case 1:

```

```

        return '<div class="weather-text">18°C</div>';
    case 2:
        return '<div class="weather-text">10°C</div>';
    case 3:
        return '<div class="weather-text">16°C</div>';
    case 4:
        return '<div class="weather-text">8°C</div>';
    case 5:
        return '<div class="weather-text">27°C</div>';
    case 6:
        return '<div class="weather-text">17°C</div>';
    default:
        return null;
    }
}
const dateHeaderTemplate = (props): JSX.Element => {
    return (<div><div>{getDateHeaderText(props.date)}</div><div
className="date-text"
    dangerouslySetInnerHTML={{ __html: getWeather(props.date)
}}></div></div>);
}
const resourceHeaderTemplate = (props): JSX.Element => {
    return (<div className="res-template-wrap"><div className={"resource-
image " + getDoctorImage(props)}></div>
    <div className="resource-detail"><div className="resource-
name">{getDoctorName(props)}</div>
    <div className="resource-
designation">{getDoctorLevel(props)}</div></div></div>);
}
const getMonthCellText = (date: Date): string => {
    if (date.getMonth() === 1 && date.getDate() === 23) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    } else if (date.getMonth() === 1 && date.getDate() === 9) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/get-together.svg" />';
    } else if (date.getMonth() === 1 && date.getDate() === 13) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    } else if (date.getMonth() === 1 && date.getDate() === 22) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/thanksgiving-day.svg"
/>';
    } else if (date.getMonth() === 1 && date.getDate() === 14) {

```

```

        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    }
    return '';
}
const getWorkCellText = (date: Date): string => {
    let weekEnds: number[] = [0, 6];
    if (weekEnds.indexOf(date.getDay()) >= 0) {
        return "<span class='caption'>Weekend</span>";
    }
    return '';
}
const cellTemplate = (props) => {
    if (props.type === "workCells") {
        return (<div className="cell-template-wrap" dangerouslySetInnerHTML={{
            __html: getWorkCellText(props.date) }}></div>);
    }
    if (props.type === "monthCells") {
        return (<div className="cell-template-wrap" dangerouslySetInnerHTML={{
            __html: getMonthCellText(props.date) }}></div>);
    }
    return (<div></div>);
}
const refreshCellTemplate = (): void => {
    scheduleObj.current.refreshTemplates("cellTemplate");
}
const refreshDateHeaderTemplate = (): void => {
    scheduleObj.current.refreshTemplates("dateHeaderTemplate");
}
const refreshEventTemplate = (): void => {
    scheduleObj.current.refreshTemplates("eventTemplate");
}
const refreshResHeaderTemplate = (): void => {
    scheduleObj.current.refreshTemplates("resourceHeaderTemplate");
}
const refreshAllTemplate = (): void => {
    scheduleObj.current.refreshTemplates();
}
return (
    <div className='schedule-control-section'>
        <div className='control-section'>
            <div className='control-wrapper'>
                <div style={{ "display": 'flex' }}>
                    <div style={{ paddingRight: '10px' }}>
                        <ButtonComponent cssClass='e-info'
onClick={refreshCellTemplate}>Refresh cellTemplate</ButtonComponent>
                    </div>
                    <div style={{ paddingRight: '10px' }}>
                        <ButtonComponent cssClass='e-info'
onClick={refreshDateHeaderTemplate}>Refresh
dateHeaderTemplate</ButtonComponent>
                    </div>
                    <div style={{ paddingRight: '10px' }}>
                        <ButtonComponent cssClass='e-info'
onClick={refreshEventTemplate}>Refresh eventTemplate</ButtonComponent>
                    </div>
                    <div style={{ paddingRight: '10px' }}>

```

```

        <ButtonComponent cssClass='e-info'
onClick={refreshResHeaderTemplate}>Refresh
resourceHeaderTemplate</ButtonComponent>
    </div>
    <div style={{ paddingRight: '10px' }}>
        <ButtonComponent cssClass='e-info'
onClick={refreshAllTemplate}>Refresh All Templates</ButtonComponent>
    </div>
</div>
<ScheduleComponent width='100%' height='650px' cssClass='schedule-
date-header-template' ref={scheduleObj}
selectedDate={new Date(2021, 1, 15)} readonly={true}
eventSettings={eventSettings}
dateHeaderTemplate={dateHeaderTemplate}
resourceHeaderTemplate={resourceHeaderTemplate} cellTemplate={cellTemplate}
group={group}>
    <ResourcesDirective>
        <ResourceDirective field='DoctorId' title='Doctor Name'
name='Doctors'
        dataSource={resourceData} textField='text' idField='id'
groupIDField='groupId' colorField='color'
        workDaysField='workDays' startHourField='startHour'
endHourField='endHour' >
    </ResourceDirective>
    </ResourcesDirective>
    <ViewsDirective>
        <ViewDirective option='Week' eventTemplate={eventTemplate} />
        <ViewDirective option='Month' />
        <ViewDirective option='TimelineMonth' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month, TimelineMonth,
Resize, DragAndDrop]} />
    </ScheduleComponent>
</div>
</div>
</div>
);
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Syncfusion React Schedule</title>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
        <meta name="description" content="Essential JS 2 for React
Components" />
        <meta name="author" content="Syncfusion" />
        <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />

```

```

<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>

<link href="index.css" rel="stylesheet" />
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>

<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>

</html>

```

Refresh Layout

In Scheduler, we can able to refresh the layout manually without re-render the DOM element by using the [refreshLayout](#) public method. The following example code explains to know how to use the refreshLayout method.

INDEX.JSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
ViewsDirective, ViewDirective } from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {

```



```

const scheduleObj = useRef(null);
const scheduleData = [{
  Id: 1,
  Subject: 'Testing',
  StartTime: new Date(2021, 10, 16, 10, 0),
  EndTime: new Date(2021, 10, 16, 12, 0),
  IsAllDay: false
}, {
  Id: 2,
  Subject: 'Vacation',
  StartTime: new Date(2021, 10, 18, 10, 0),
  EndTime: new Date(2021, 10, 18, 12, 0),
  IsAllDay: false
}];
const eventSettings = { dataSource: scheduleData };
const onRefreshLayout = () => {
  scheduleObj.current.refreshLayout();
}
return (<div>
  <ButtonComponent onClick={onRefreshLayout}>Refresh
Layout</ButtonComponent> <ScheduleComponent ref={scheduleObj} width='100%'
height='550px' selectedDate={new Date(2021, 10, 15)}
eventSettings={eventSettings}>
  <ViewsDirective>
    <ViewDirective option='Day' />
    <ViewDirective option='Week' />
    <ViewDirective option='WorkWeek' />
    <ViewDirective option='Month' />
  </ViewsDirective>
  <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>
</div>);
}
;
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.TSX

```

import { useRef } from 'react';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import {
  ScheduleComponent, Day, Week, WorkWeek, Month, Inject,
  ViewsDirective, ViewDirective, EventSettingsModel
} from '@syncfusion/ej2-react-schedule';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const App = () => {
  const scheduleObj = useRef<ScheduleComponent>(null);
  const scheduleData: Object[] = [{
    Id: 1,
    Subject: 'Testing',
    StartTime: new Date(2021, 10, 16, 10, 0),
    EndTime: new Date(2021, 10, 16, 12, 0),
    IsAllDay: false
  }, {

```

```

    Id: 2,
    Subject: 'Vacation',
    StartTime: new Date(2021, 10, 18, 10, 0),
    EndTime: new Date(2021, 10, 18, 12, 0),
    IsAllDay: false
  }];
  const eventSettings: EventSettingsModel = { dataSource: scheduleData };
  const onRefreshLayout = (): void => {
    scheduleObj.current.refreshLayout();
  }
  return (<div>
    <ButtonComponent onClick={onRefreshLayout}>Refresh
  Layout</ButtonComponent> <ScheduleComponent ref={scheduleObj} width='100%'
  height='550px' selectedDate=
    {new Date(2021, 10, 15)} eventSettings={eventSettings}>
    <ViewsDirective>
      <ViewDirective option='Day' />
      <ViewDirective option='Week' />
      <ViewDirective option='WorkWeek' />
      <ViewDirective option='Month' />
    </ViewsDirective>
    <Inject services={[Day, Week, WorkWeek, Month]} />
  </ScheduleComponent>
</div>)
};
const root = ReactDOM.createRoot(document.getElementById('schedule'));
root.render(<App />);

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Syncfusion React Schedule</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
  />
    <meta name="description" content="Essential JS 2 for React
Components" />
    <meta name="author" content="Syncfusion" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-
base/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
buttons/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
dropdowns/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
inputs/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
navigations/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
popups/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
schedule/styles/material.css" rel="stylesheet" />

```

```
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.38/system.js"></scr
ipt>
<script src="systemjs.config.js"></script>
<style>
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
</head>

<body>
    <div id='schedule'>
        <div id='loader'>Loading....</div>
    </div>
</body>

</html>
```

Sidebar

Getting Started

The following section explains the required steps to build the simple **Sidebar** component with its basic usage in step by step procedure.

Dependencies

The following list of dependencies are required to use the Sidebar component in your application.

```
`javascript
|-- @syncfusion/ej2-react-navigations
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-react-base
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
```

,

Installation and configuration

You can use [create-react-app](#) to setup the applications.

To install `create-react-app` run the following command.

,

```
npm install -g create-react-app
```

,

- To setup basic `React` sample use following commands.

```
<div class='tsx'>
```

,

```
create-react-app quickstart --scripts-version=react-scripts-ts
```

```
cd quickstart
```

,

```
</div>
```

```
<div class='jsx'>
```

,

```
create-react-app quickstart
```

```
cd quickstart
```

,

```
</div>
```

Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](#) public registry. You can choose the component that you want to install. For this application, we are going to use `Sidebar` component.

To install Sidebar component, use the following command

```
`bash
```

```
npm install @syncfusion/ej2-react-navigations --save
```

,

Adding Style sheet to the Application

To render the Sidebar component, need to import Sidebar and its dependent component's styles as given below in `App.css`.

```
`css
```

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-react-navigations/styles/material.css";
```

,

Note: If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

Adding Sidebar component to the Application

Sidebar can be initialized using the `<SidebarComponent>` tag, it's used to render Sidebar as it contains primary content aside from the main content. The immediate sibling element of the Sidebar will be considered as the main content.

- To include the Sidebar component in application import the `SidebarComponent` from `ej2-react-navigations` package in `App.tsx`.
- Then add the Sidebar component as shown in below code example.

[src/App.tsx]

`ts

```
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
```

```
import * as React from 'react';
```

```
function App() {
```

```
  return (
```

```
    <div className="control-section">
```

```
      <div id="wrapper">
```

```
        <SidebarComponent id="default-sidebar">
```

```
          <div className="title"> Sidebar content</div>
```

```
        </SidebarComponent>
```

```
        <div className="content">
```

```
          <div className="title">Main content</div>
```

```
          <div className="sub-title"> content goes here</div>
```

```
        </div>
```

```
      </div>
```

```
    </div>
```

```
  )
```

```
}
```

```
export default App;
```

,

`ts

```
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
```

```
import * as React from 'react';
```

```
function App() {
  return (<div className="control-section">
    <div id="wrapper">
      <SidebarComponent id="default-sidebar">
        <div className="title"> Sidebar content</div>
      </SidebarComponent>
      <div className="content">
        <div className="title">Main content</div>
        <div className="sub-title"> content goes here</div>
      </div>
    </div>
  </div>);
}
export default App;
```

Run the application

Now run the `npm start` command in the console, it will run your application and open the browser window.

```
npm start
```

The following sample, shows the basic Sidebar component.

APP.JSX

```
{% raw %}
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
  let sidebarObj;
  function onCreate() {
    sidebarObj.element.style.visibility = '';
  }
  return (<div className="control-section">
    <div id="wrapper">
      { /* Initializing the Sidebar component */ }
      <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarObj = Sidebar} created={onCreate} style={{ visibility: "hidden" }}>
        <div className="title"> Sidebar content</div>
      </SidebarComponent>
      <div className="content">
        <div className="title">Main content</div>
        <div className="sub-title"> content goes here</div>
      </div>
    </div>
  </div>);
}
```

```

        </div>
      </div>
    </div>);
  }
  export default App;
  {% endraw %}

```

APP.TSX

```

{% raw %}
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
  let sidebarObj: SidebarComponent;
  function onCreate():void {
    sidebarObj.element.style.visibility='';
  }
  return (
    <div className="control-section">
      <div id="wrapper">
        {/* Initializing the Sidebar component */}
        <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarObj = Sidebar as SidebarComponent} created={onCreate}
style={{visibility:"hidden"}}>
          <div className="title"> Sidebar content</div>
        </SidebarComponent>
        <div className="content">
          <div className="title">Main content</div>
          <div className="sub-title"> content goes here</div>
        </div>
      </div>
    </div>
  )
}
export default App;
{% endraw %}

```

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

Enable backdrop

Enabling the [showBackdrop](#) in the Sidebar component will prevent the main content from user interactions.

Here, DOM elements will not get changed. It only closes the main content by covering with a black backdrop overlay and focuses the Sidebar in the screen. Sidebar can be rendered with specific width by setting `width` property

Note: To achieve a proper **backdrop**, we suggest that you create a wrapper parent container for the div block in which you intend to enable the backdrop. Set the class name of this parent container as the **target** for the Sidebar. Alternatively, you can place an empty div container after the target container.

APP.JSX

```
{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
  let sidebarObj;
  let type = "Push";
  let showBackdrop = true;
  function onCreate() {
    sidebarObj.element.style.visibility = '';
  }
  // Toggle(Open/Close) the Sidebar
  function toggleClick() {
    sidebarObj.toggle();
  }
  // Close the Sidebar
  function closeClick() {
    sidebarObj.hide();
  }
  return (<div className="control-section">
    <div id="wrapper">
      /* Initializing the Sidebar component */
      <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarObj = Sidebar} style={{ visibility: "hidden" }} created={onCreate}
showBackdrop={showBackdrop} type={type} width="280px">
        <div className="title"> Sidebar content</div>
        <div className="sub-title">
          Click the button to close the Sidebar.
        </div>
        <div className="center-align">
          <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
        </div>
      </SidebarComponent>
    <div>
      <div className="title">Main content</div>
      <div className="sub-title"> Click the button to
open/close the Sidebar.</div>
      <div className="center-align">
        <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Toggle Sidebar</ButtonComponent>
      </div>
    </div>
  </div>);
}
export default App;
```



```
{% endraw %}
```

APP.TSX

```
{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent, SidebarType } from '@syncfusion/ej2-react-
navigations';
import * as React from 'react';
function App() {
    let sidebarObj: SidebarComponent;
    let type: SidebarType = "Push";
    let showBackdrop: boolean = true;
    function onCreate(): void {
        sidebarObj.element.style.visibility='';
    }
    // Toggle(Open/Close) the Sidebar
    function toggleClick(): void {
        sidebarObj.toggle();
    }
    // Close the Sidebar
    function closeClick(): void {
        sidebarObj.hide();
    }
    return (
        <div className="control-section">
            <div id="wrapper">
                /* Initializing the Sidebar component */
                <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarObj = Sidebar as SidebarComponent} style={{visibility:"hidden"}}
created={onCreate} showBackdrop={showBackdrop} type={type} width="280px">
                    <div className="title"> Sidebar content</div>
                    <div className="sub-title">
                        Click the button to close the Sidebar.
                    </div>
                    <div className="center-align">
                        <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
                    </div>
                </SidebarComponent>
                <div>
                    <div className="title">Main content</div>
                    <div className="sub-title"> Click the button to
open/close the Sidebar.</div>
                    <div className="center-align">
                        <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Toggle Sidebar</ButtonComponent>
                    </div>
                </div>
            </div>
        </div>
    )
}
export default App;
{% endraw %}
```

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));
```

Position

Positioning the Sidebar to the right or left of the main content can be achieved by using the [position](#) property. If the position is not set, the Sidebar will expand from the left to the body element. [enablePersistence](#) will persist the component's state between page reloads. [change](#) event will be triggered when the state(expand/collapse) of the component is changed.

APP.JSX

```
{% raw %}
import { ButtonComponent, RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
    let sidebarObj;
    let btnObj;
    let type = "Push";
    function onCreate() {
        sidebarObj.element.style.visibility = '';
    }
    // Toggle button click event handler
    function btnClick() {
        if (btnObj.element.classList.contains('e-active')) {
            btnObj.content = 'Close';
            btnObj.iconCss = 'e-icons burg-icon';
            sidebarObj.show();
        }
        else {
            btnObj.content = 'Open';
            btnObj.iconCss = 'e-icons burg-icon';
            sidebarObj.hide();
        }
    }
    // Toggle(Open/Close) the Sidebar
    function toggleClick() {
        sidebarObj.toggle();
    }
    // Close the Sidebar
    function closeClick() {
        sidebarObj.hide();
        btnObj.content = 'Open';
    }
}
```

```

    }
    // function to handle the CheckBox change event
    function onChange(args) {
        sidebarObj.position = (args.event.target.id === "left") ? "Left" :
"Right";
    }
    return (<div className="control-section">
        <div id="wrapper">
            /* Initializing the Sidebar component */
            <SidebarComponent id="default-sidebar"
enablePersistence={true} ref={Sidebar => sidebarObj = Sidebar} style={{
visibility: "hidden" }} created={onCreate} type={type} width="280px">
                <div className="title"> Sidebar content</div>
                <div className="sub-title"> Click the button to close the
Sidebar. </div>
                <div className="center-align">
                    <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
                </div>
            </SidebarComponent>
            <div id="head">
                <ButtonComponent id="toggle" ref={(scope) => { btnObj =
scope; }} iconCss='e-icons burg-icon' isToggle={true} onClick={btnClick}
className="e-btn e-info">Open</ButtonComponent>
            </div>
            <div id="maincontent" className="content">
                <div>
                    <div className="title">Main content</div>
                    <div className="radiobutton">
                        <div className='row'>
                            <RadioButtonComponent id="left"
checked={true} label='Left' name='position' change={onChange}/>
                        </div>
                        <div className='row'>
                            <RadioButtonComponent id="right"
label='Right' name='position' change={onChange}/>
                        </div>
                    </div>
                </div>
            </div>
        </div>);
    }
    export default App;
    {% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent, ChangeEventArgs, RadioButtonComponent } from
'@syncfusion/ej2-react-buttons';
import { SidebarComponent, SidebarType } from '@syncfusion/ej2-react-
navigations';
import * as React from 'react';
function App() {
    let sidebarObj: SidebarComponent;

```

```

let btnObj: ButtonComponent;
let type: SidebarType = "Push";
function onCreate(): void {
    sidebarObj.element.style.visibility='';
}
// Toggle button click event handler
function btnClick(): void {
    if (btnObj.element.classList.contains('e-active')) {
        btnObj.content = 'Close';
        btnObj.iconCss = 'e-icons burg-icon';
        sidebarObj.show();
    } else {
        btnObj.content = 'Open';
        btnObj.iconCss = 'e-icons burg-icon';
        sidebarObj.hide();
    }
}
// Toggle(Open/Close) the Sidebar
function toggleClick(): void {
    sidebarObj.toggle();
}
// Close the Sidebar
function closeClick(): void {
    sidebarObj.hide();
    btnObj.content = 'Open';
}
// function to handle the CheckBox change event
function onChange(args: ChangeEventArgs): void {
    sidebarObj.position = ((args as any).event.target.id === "left") ?
"Left" : "Right";
}
return (
    <div className="control-section">
        <div id="wrapper">
            /* Initializing the Sidebar component */
            <SidebarComponent id="default-sidebar"
enablePersistence={true} ref={Sidebar => sidebarObj = Sidebar as
SidebarComponent} style={{visibility:"hidden"}} created={onCreate}
type={type} width="280px">
                <div className="title"> Sidebar content</div>
                <div className="sub-title"> Click the button to close the
Sidebar. </div>
                <div className="center-align">
                    <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
                </div>
            </SidebarComponent>
            <div id="head">
                <ButtonComponent id="toggle" ref={(scope) => { btnObj =
scope as ButtonComponent; }} iconCss='e-icons burg-icon'
isToggle={true} onClick={btnClick} className="e-btn
e-info">Open</ButtonComponent>
            </div>
            <div id="maincontent" className="content">
                <div>
                    <div className="title">Main content</div>
                    <div className="radiobutton" >

```

```

        <div className='row'>
            <RadioButtonComponent id="left"
checked={true} label='Left' name='position' change={onChange} />
        </div>
        <div className='row'>
            <RadioButtonComponent id="right"
label='Right' name='position' change={onChange} />
        </div>
    </div>
</div>
</div>
</div>
</div>
</div>
    )
}
export default App;
{% endraw %}

```

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

Animate

Animation transitions can be set while expanding or collapsing the Sidebar using the [animate](#) property. By default, [animate](#) property is set to true. [enableRTL](#) will display the sidebar in the right-to-left direction.

APP.JSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
    let sidebarObj;
    let type = "Push";
    function onCreate() {
        sidebarObj.element.style.visibility = '';
    }
    // Toggle(Open/Close) the Sidebar
    function toggleClick() {
        sidebarObj.toggle();
    }
    // Close the Sidebar
    function closeClick() {

```

```

        sidebarObj.hide();
    }
    return (<div className="control-section">
        <div id="wrapper">
            /* Initializing the Sidebar component */
            <SidebarComponent id="default-sidebar" animate={false}
enableRtl={true} type={type} ref={Sidebar => sidebarObj = Sidebar}
created={onCreate} style={{ visibility: "hidden" }}>
                <div className="title"> Sidebar content</div>
                <div className="sub-title">
                    Click the button to close the Sidebar
                </div>
                <div className="center-align">
                    <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
                </div>
            </SidebarComponent>
            <div>
                <div className="title">Main content</div>
                <div className="sub-title"> Click the button to
open/close the Sidebar.</div>
                <div className="center-align">
                    <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Toggle Sidebar</ButtonComponent>
                </div>
            </div>
        </div>);
    }
    export default App;
    {% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent, SidebarType } from '@syncfusion/ej2-react-
navigations';
import * as React from 'react';
function App() {
    let sidebarObj: SidebarComponent;
    let type: SidebarType = "Push";
    function onCreate():void {
        sidebarObj.element.style.visibility='';
    }
    // Toggle(Open/Close) the Sidebar
    function toggleClick(): void {
        sidebarObj.toggle();
    }
    // Close the Sidebar
    function closeClick(): void {
        sidebarObj.hide();
    }
    return (
        <div className="control-section">
            <div id="wrapper">

```

```

        /* Initializing the Sidebar component */
        <SidebarComponent id="default-sidebar" animate={false}
enableRtl={true} type={type} ref={Sidebar => sidebarObj = Sidebar as
SidebarComponent} created={onCreate} style={{visibility:"hidden"}} >
        <div className="title"> Sidebar content</div>
        <div className="sub-title">
            Click the button to close the Sidebar
        </div>
        <div className="center-align">
            <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
        </div>
    </SidebarComponent>
    <div>
        <div className="title">Main content</div>
        <div className="sub-title"> Click the button to
open/close the Sidebar.</div>
        <div className="center-align">
            <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Toggle Sidebar</ButtonComponent>
        </div>
    </div>
</div>
)
}
export default App;
{% endraw %}

```

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

Close on document click

Sidebar can be closed on document click by setting [closeOnDocumentClick](#) to true. If this property is not set, the Sidebar will not close on document click since its default value is false. Sidebar can be kept opened during rendering using [isOpen](#) property.

APP.JSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {

```

```

let sidebarObj;
let type = "Push";
function onCreate() {
    sidebarObj.element.style.visibility = '';
}
// Toggle(Open/Close) the Sidebar
function toggleClick() {
    sidebarObj.show();
}
return (<div className="control-section">
    <div id="wrapper">
        /* Initializing the Sidebar component */
        <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarObj = Sidebar} closeOnDocumentClick={true} type={type}
created={onCreate} isOpen={true} style={{ visibility: "hidden" }}>
            <div className="title"> Sidebar content</div>
        </SidebarComponent>
        <div>
            <div className="title">Main content</div>
            <div className="sub-title"> Click the button to open the
Sidebar.</div>
            <div className="center-align">
                <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Open Sidebar</ButtonComponent>
            </div>
        </div>
    </div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent, SidebarType } from '@syncfusion/ej2-react-
navigations';
import * as React from 'react';
function App() {
    let sidebarObj: SidebarComponent;
    let type: SidebarType = "Push";
    function onCreate():void {
        sidebarObj.element.style.visibility='';
    }
    // Toggle(Open/Close) the Sidebar
    function toggleClick(): void {
        sidebarObj.show();
    }
    return (
        <div className="control-section">
            <div id="wrapper">
                /* Initializing the Sidebar component */
                <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarObj = Sidebar as SidebarComponent} closeOnDocumentClick={true}
type={type} created={onCreate} isOpen={true} style={{visibility:"hidden"}} >

```



```

        <div className="title"> Sidebar content</div>
      </SidebarComponent>
    <div>
      <div className="title">Main content</div>
      <div className="sub-title"> Click the button to open the
Sidebar.</div>

      <div className="center-align">
        <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Open Sidebar</ButtonComponent>
      </div>
    </div>
  </div>
</div>
)
}
export default App;
{% endraw %}

```

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

Enable gestures

Expand or collapse the Sidebar while swiping in touch devices using `enableGestures` property. By default, `enableGestures` is set to true.

APP.JSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
  let sidebarObj;
  let type = "Push";
  function onCreate() {
    sidebarObj.element.style.visibility = '';
  }
  // Toggle(Open/Close) the Sidebar
  function toggleClick() {
    sidebarObj.toggle();
  }
  // Close the Sidebar
  function closeClick() {
    sidebarObj.hide();
  }
}

```

```

    }
    return (<div className="control-section">
      <div id="wrapper">
        /* Initializing the Sidebar component */
        <SidebarComponent id="default-sidebar" enableGestures={false}
type={type} ref={Sidebar => sidebarObj = Sidebar} created={onCreate} style={{
visibility: "hidden" }}>
          <div className="title"> Sidebar content</div>
          <div className="sub-title">
            Click the button to close the Sidebar.
          </div>
          <div className="center-align">
            <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
          </div>
        </SidebarComponent>
      <div>
        <div className="title">Main content</div>
        <div className="sub-title"> Click the button to
open/close the Sidebar.</div>
        <div className="center-align">
          <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Toggle Sidebar</ButtonComponent>
        </div>
      </div>
    </div>);
  }
  export default App;
  {% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent, SidebarType } from '@syncfusion/ej2-react-
navigations';
import * as React from 'react';
function App() {
  let sidebarObj: SidebarComponent;
  let type: SidebarType = "Push";
  function onCreate():void {
    sidebarObj.element.style.visibility='';
  }
  // Toggle(Open/Close) the Sidebar
  function toggleClick(): void {
    sidebarObj.toggle();
  }
  // Close the Sidebar
  function closeClick(): void {
    sidebarObj.hide();
  }
  return (
    <div className="control-section">
      <div id="wrapper">
        /* Initializing the Sidebar component */

```

```

        <SidebarComponent id="default-sidebar" enableGestures={false}
type={type} ref={Sidebar => sidebarObj = Sidebar as SidebarComponent}
created={onCreate} style={{visibility:"hidden"}} >
        <div className="title"> Sidebar content</div>
        <div className="sub-title">
            Click the button to close the Sidebar.
        </div>
        <div className="center-align">
            <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
        </div>
    </SidebarComponent>
    <div>
        <div className="title">Main content</div>
        <div className="sub-title"> Click the button to
open/close the Sidebar.</div>
        <div className="center-align">
            <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Toggle Sidebar</ButtonComponent>
        </div>
    </div>
</div>
)
}
export default App;
{% enddraw %}

```

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

See Also

- [Sidebar with navigation menu](#)
- [Sidebar responsive panel](#)
- [Sidebar with listview](#)

Custom context in React Sidebar component

By default, Sidebar initializes target to the body element. Using the [target](#) property, set target element to initialize the Sidebar inside any HTML element apart from the body element.

If required, `zIndex` can be set when sidebar act as overlay type.

APP.JSX

```
{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
    let sidebarObj;
    let btnObj;
    let target = '.content';
    function onCreate() {
        sidebarObj.element.style.visibility = '';
    }
    // Toggle(Open/Close) the Sidebar
    function toggleClick() {
        sidebarObj.toggle();
    }
    // Close the Sidebar
    function closeClick() {
        sidebarObj.hide();
        btnObj.content = 'Open';
    }
    // Toggle button click event handler
    function btnClick() {
        if (btnObj.element.classList.contains('e-active')) {
            btnObj.content = 'Close';
            sidebarObj.show();
        }
        else {
            btnObj.content = 'Open';
            sidebarObj.hide();
        }
    }
    return (<div className="control-section">
        <div id="wrapper">
            /* Initializing the Sidebar component */
            <SidebarComponent id="default-sidebar" type="Push"
target={target} style={{ visibility: "hidden" }} created={onCreate}
ref={sidebar => sidebarObj = sidebar}>
                <div className="title"> Sidebar content</div>
                <div className="sub-title">
                    Click the button to close the Sidebar.
                </div>
                <div className="center-align">
                    <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
                </div>
            </SidebarComponent>
            <div id="head">
                <ButtonComponent id="toggle" ref={(scope) => { btnObj =
scope; }} iconCss='e-icons burg-icon' isToggle={true} onClick={btnClick}
className="e-btn e-info">Open</ButtonComponent>
            </div>
            <div id="maincontent" className="content">
                <div>
                    <div className="title">Main content</div>
                    <div className="center-align">

```

```

        Content goes here.
      </div>
    </div>
  </div>
</div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
  let sidebarObj: SidebarComponent;
  let btnObj: ButtonComponent;
  let target: string = '.content';
  function onCreate(): void {
    sidebarObj.element.style.visibility='';
  }
  // Toggle(Open/Close) the Sidebar
  function toggleClick(): void {
    sidebarObj.toggle();
  }
  // Close the Sidebar
  function closeClick(): void {
    sidebarObj.hide();
    btnObj.content = 'Open';
  }
  // Toggle button click event handler
  function btnClick(): void {
    if (btnObj.element.classList.contains('e-active')) {
      btnObj.content = 'Close';
      sidebarObj.show();
    } else {
      btnObj.content = 'Open';
      sidebarObj.hide();
    }
  }
  return (
    <div className="control-section">
      <div id="wrapper">
        { /* Initializing the Sidebar component */ }
        <SidebarComponent id="default-sidebar" type="Push"
target={target} style={{visibility:"hidden"}} created={onCreate} ref={sidebar
=> sidebarObj = sidebar as SidebarComponent}>
          <div className="title"> Sidebar content</div>
          <div className="sub-title">
            Click the button to close the Sidebar.
          </div>
          <div className="center-align">
            <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>

```

```

        </div>
      </SidebarComponent>
      <div id="head">
        <ButtonComponent id="toggle" ref={(scope) => { btnObj =
scope as ButtonComponent }} iconCss='e-icons burg-icon'
        isToggle={true} onClick={btnClick} className="e-btn
e-info">Open</ButtonComponent>
      </div>
      <div id="maincontent" className="content">
        <div>
          <div className="title">Main content</div>
          <div className="center-align" >
            Content goes here.
          </div>
        </div>
      </div>
    </div>
  </div>
)
}
export default App;
{% endraw %}

```

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

Variations in React Sidebar component

The Sidebar component's expand behavior can be modified based on the purpose of use.

- Expanding types of the Sidebar

Expanding types of Sidebar

Sidebar can be set to initialize based on four different types that are consistent with the main component as explained below. When `dataBind` is invoked, this applies the pending property changes immediately to the component.

| Item | Description |

|-----|-----|

| [Over](#) | Sidebar floats over the main content area. |

| [Push](#) | Sidebar pushes the main content area to appear side-by-side, and shrinks the main content within the screen width. |

| [Slide](#) | Sidebar translates the x and y positions of main content area based on the Sidebar width. The main content area will not be adjusted within the screen width. |

| [Auto](#) | Sidebar with **Over** type in mobile resolution, and **Push** type in other higher resolutions. |

Auto is the default expand mode.

In the following sample, the types of Sidebar are demonstrated.

APP.JSX

```
{% raw %}
import { ButtonComponent, RadioButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
    let sidebarObj;
    let btnObj;
    let type = "Push";
    let target = '.content';
    function onCreate() {
        sidebarObj.element.style.visibility = '';
    }
    // Toggle button click event handler
    function btnClick() {
        if (btnObj.content === 'Open') {
            btnObj.content = 'Close';
            sidebarObj.show();
        }
        else {
            btnObj.content = 'Open';
            sidebarObj.hide();
        }
    }
    //change the toggle button state
    function changestate() {
        if (sidebarObj.type === "Auto") {
            btnObj.content = 'Close';
            sidebarObj.show();
        }
        else {
            btnObj.content = 'Open';
            sidebarObj.hide();
        }
    }
    // function to handle the CheckBox change event
    function onChange(args) {
        if (args.event.target.id === "over") {
            sidebarObj.type = "Over";
        }
        else if (args.event.target.id === "push") {
            sidebarObj.type = "Push";
        }
        else if (args.event.target.id === "slide") {
```

```

        sidebarObj.type = "Slide";
    }
    else {
        sidebarObj.type = "Auto";
    }
    changestate();
    sidebarObj.dataBind();
}
// Toggle(Open/Close) the Sidebar
function toggleClick() {
    sidebarObj.toggle();
}
// Close the Sidebar
function closeClick() {
    sidebarObj.hide();
    btnObj.content = 'Open';
}
return (<div className="control-section">
    <div id="wrapper">
        /* Initializing the Sidebar component */
        <SidebarComponent id="default-sidebar" target={target}
type={type} created={onCreate} style={{ visibility: "hidden" }} ref={sidebar
=> sidebarObj = sidebar}>
            <div className="title"> Sidebar content</div>
            <div className="sub-title"> Click the button to close the
Sidebar. </div>
            <div className="center-align">
                <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
            </div>
            </SidebarComponent>
            <div id="head">
                <ButtonComponent id="toggle" ref={(scope) => { btnObj =
scope; }} iconCss='e-icons burg-icon' isToggle={true} onClick={btnClick}
className="e-btn e-info" content='Open' />
            </div>
            <div id="maincontent" className="content">
                <div>
                    <div className="title">Main content</div>
                    <div className="sub-title"> Click the button to
open/close the Sidebar.</div>
                    <div className="radiobutton">
                        <div className='row'>
                            <RadioButtonComponent id="over" label='Over'
name='type' change={onChange}/>
                        </div>
                        <div className='row'>
                            <RadioButtonComponent id="push"
checked={true} label='Push' name='type' change={onChange}/>
                        </div>
                        <div className='row'>
                            <RadioButtonComponent id="slide"
label='Slide' name='type' change={onChange}/>
                        </div>
                        <div className='row'>
                            <RadioButtonComponent id="auto" label='Auto'
name='type' change={onChange}/>

```



```

        </div>
      </div>
    </div>
  </div>
</div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent, ChangeEventArgs ,RadioButtonComponent } from
 '@syncfusion/ej2-react-buttons';
import { SidebarComponent, SidebarType } from '@syncfusion/ej2-react-
 navigations';
import * as React from 'react';
function App() {
  let sidebarObj: SidebarComponent;
  let btnObj: ButtonComponent;
  let type: SidebarType = "Push";
  let target: string = '.content';
  function onCreate(): void {
    sidebarObj.element.style.visibility='';
  }
  // Toggle button click event handler
  function btnClick(): void {
    if (btnObj.content === 'Open') {
      btnObj.content = 'Close';
      sidebarObj.show();
    } else {
      btnObj.content = 'Open';
      sidebarObj.hide();
    }
  }
  //change the toggle button state
  function changestate() {
    if (sidebarObj.type === "Auto") {
      btnObj.content = 'Close';
      sidebarObj.show();
    }
    else {
      btnObj.content = 'Open';
      sidebarObj.hide();
    }
  }
  // function to handle the CheckBox change event
  function onChange(args: ChangeEventArgs): void {
    if ((args as any).event.target.id === "over") {
      sidebarObj.type = "Over";
    } else if ((args as any).event.target.id === "push") {
      sidebarObj.type = "Push";
    } else if ((args as any).event.target.id === "slide") {
      sidebarObj.type = "Slide";
    } else {

```

```

        sidebarObj.type = "Auto";
    }
    changestate();
    sidebarObj.dataBind();
}
// Toggle (Open/Close) the Sidebar
function toggleClick(): void {
    sidebarObj.toggle();
}
// Close the Sidebar
function closeClick(): void {
    sidebarObj.hide();
    btnObj.content = 'Open';
}
return (
    <div className="control-section">
        <div id="wrapper">
            /* Initializing the Sidebar component */
            <SidebarComponent id="default-sidebar" target={target}
type={type} created={onCreate} style={{visibility:"hidden"}} ref={sidebar =>
sidebarObj = sidebar as SidebarComponent}>
                <div className="title"> Sidebar content</div>
                <div className="sub-title"> Click the button to close the
Sidebar. </div>
                <div className="center-align">
                    <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
                </div>
            </SidebarComponent>
            <div id="head">
                <ButtonComponent id="toggle" ref={(scope) => { btnObj =
scope as ButtonComponent }} iconCss='e-icons burg-icon'
isToggle={true} onClick={ btnClick} className="e-btn e-
info" content='Open' />
            </div>
            <div id="maincontent" className="content">
                <div>
                    <div className="title">Main content</div>
                    <div className="sub-title"> Click the button to
open/close the Sidebar.</div>
                    <div className="radiobutton" >
                        <div className='row'>
                            <RadioButtonComponent id="over" label='Over'
name='type' change={onChange}/>
                        </div>
                        <div className='row'>
                            <RadioButtonComponent id="push"
checked={true} label='Push' name='type' change={onChange} />
                        </div>
                        <div className='row'>
                            <RadioButtonComponent id="slide"
label='Slide' name='type' change={onChange} />
                        </div>
                        <div className='row'>
                            <RadioButtonComponent id="auto" label='Auto'
name='type' change={onChange} />
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

```

```

        </div>
      </div>
    </div>
  </div>
)
}
export default App;
{% endraw %}

```

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

See Also

- [How to add sidebar with custom animation](#)
- [How to add multiple sidebar](#)

Auto close in React Sidebar component

Sidebar often behaves differently on mobile display and differently on desktop display. It has an effective feature that offers to set it in opened or closed state depending on the specified resolution. This is achieved through [mediaQuery](#) property that allows you to set the Sidebar in an expanded state or collapsed state only in user-defined resolution.

APP.JSX

```

{% raw %}
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
  let sidebarobj;
  let mediaQueryState = { mediaQuery: window.matchMedia('(min-width: 600px)') };
  function onCreate() {
    sidebarobj.element.style.visibility = '';
  }
  return (
    <div className="control-section">
      <div id="wrapper">
        <div className="col-lg-12 col-sm-12 col-md-12">
          /* Initializing the Sidebar component */

```

```

        <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarobj = Sidebar} created={onCreate} style={{ visibility: "hidden" }}
mediaQuery={mediaQueryState.mediaQuery} width="248px">
            <div className="title"> Sidebar content</div>
        </SidebarComponent>
    </div>
    <div>
        <div className="title">Main content</div>
        <div className="sub-title"> * Sidebar will collapse
in mobile mode automatically</div>
    </div>
</div>
</div>
</div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
    let sidebarobj: SidebarComponent;
    let mediaQueryState={ mediaQuery: window.matchMedia('(min-width:
600px)')}
    function onCreate() {
        sidebarobj.element.style.visibility='';
    }
    return (
        <div className="control-section">
            <div id="wrapper">
                <div className="col-lg-12 col-sm-12 col-md-12">
                    /* Initializing the Sidebar component */
                    <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarobj = Sidebar as SidebarComponent} created={onCreate}
style={{visibility:"hidden"}} mediaQuery={mediaQueryState.mediaQuery}
width="248px">
                        <div className="title"> Sidebar content</div>
                    </SidebarComponent>
                </div>
                <div>
                    <div className="title">Main content</div>
                    <div className="sub-title"> * Sidebar will collapse
in mobile mode automatically</div>
                </div>
            </div>
        </div>
    )
}
export default App;
{% endraw %}

```

INDEX.JSX

```
import * as React from 'react';
```

```
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));
```

- In this sample, the Sidebar will be in expanded state only in resolution below 400px.

APP.JSX

```
{% raw %}
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
  let sidebarobj;
  let mediaQueryState = { mediaQuery: window.matchMedia('(max-width:
400px)') };
  function onCreate() {
    sidebarobj.element.style.visibility = '';
  }
  return (<div className="control-section">
    <div id="wrapper">
      <div className="col-lg-12 col-sm-12 col-md-12">
        {/* Initializing the Sidebar component */}
        <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarobj = Sidebar} mediaQuery={mediaQueryState.mediaQuery} style={{
visibility: "hidden" }} created={onCreate} width="248px">
          <div className="title"> Sidebar content</div>
        </SidebarComponent>
        <div>
          <div className="title">Main content</div>
          <div className="sub-title"> * Sidebar will expanded
in mobile mode automatically</div>
        </div>
      </div>
    </div>
  </div>);
}
export default App;
{% endraw %}
```

APP.TSX

```
{% raw %}
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
  let sidebarobj: SidebarComponent;
```

```

    let mediaQueryState={ mediaQuery: window.matchMedia('(max-width:
400px)')});
    function onCreate(): void {
        sidebarobj.element.style.visibility='';
    }
    return (
        <div className="control-section">
            <div id="wrapper">
                <div className="col-lg-12 col-sm-12 col-md-12">
                    /* Initializing the Sidebar component */
                    <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarobj = Sidebar as SidebarComponent}
mediaQuery={mediaQueryState.mediaQuery} style={{visibility:"hidden"}}
created={onCreate} width="248px">
                        <div className="title"> Sidebar content</div>
                    </SidebarComponent>
                    <div>
                        <div className="title">Main content</div>
                        <div className="sub-title"> * Sidebar will expanded
in mobile mode automatically</div>
                    </div>
                </div>
            </div>
        </div>
    )
}
export default App;
{% endraw %}

```

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

Docking sidebar in React Sidebar component

Dock state of the Sidebar reserves some space on the page that always remains in a visible state when the Sidebar is collapsed. It is used to show the short term of a content like icons alone instead of lengthy text. To achieve this, set [enableDock](#) as true along with required [dockSize](#).

In the following sample, the list item has icon with text representation. On dock state, only icons in list will be visible to represent the hint of the hidden text content.

APP.JSX

```

import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';

```

```
function App() {
  let dockBar;
  // Toggle(Open/Close) the Sidebar
  function toggleClick() {
    dockBar.toggle();
  }
  return (<div className="control-section">
    <div id="wrapper">
      {/* Initializing the Sidebar component */}
      <SidebarComponent id="dockSidebar" ref={Sidebar => dockBar =
Sidebar} enableDock={true} dockSize="72px" width="220px">
        <div className="dock">
          <ul>
            <li className="sidebar-item" id="toggle"
onClick={toggleClick}>
              <span className="e-icons expand"/>
              <span className="e-text"
title="menu">Menu</span>
            </li>
            <li className="sidebar-item">
              <span className="e-icons home"/>
              <span className="e-text"
title="home">Home</span>
            </li>
            <li className="sidebar-item">
              <span className="e-icons profile"/>
              <span className="e-text"
title="profile">Profile</span>
            </li>
            <li className="sidebar-item">
              <span className="e-icons info"/>
              <span className="e-text"
title="info">Info</span>
            </li>
            <li className="sidebar-item">
              <span className="e-icons settings"/>
              <span className="e-text"
title="settings">Settings</span>
            </li>
          </ul>
        </div>
      </SidebarComponent>
      <div id="main-content container-fluid col-md-12 ">
        <div className="title">Main content</div>
        <div className="sub-title"> Click the expand icon to open
and collapse icons to close the Sidebar</div>
      </div>
    </div>
  </div>);
}
export default App;
```

APP.TSX

```
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
```

```

function App() {
  let dockBar: SidebarComponent;
  // Toggle(Open/Close) the Sidebar
  function toggleClick() {
    dockBar.toggle();
  }
  return (
    <div className="control-section">
      <div id="wrapper">
        {/* Initializing the Sidebar component */}
        <SidebarComponent id="dockSidebar" ref={Sidebar => dockBar =
Sidebar as SidebarComponent} enableDock={true} dockSize="72px" width="220px"
>
          <div className="dock">
            <ul>
              <li className="sidebar-item" id="toggle"
onClick={toggleClick} >
                <span className="e-icons expand"/>
                <span className="e-text"
title="menu">Menu</span>
              </li>
              <li className="sidebar-item">
                <span className="e-icons home"/>
                <span className="e-text"
title="home">Home</span>
              </li>
              <li className="sidebar-item">
                <span className="e-icons profile"/>
                <span className="e-text"
title="profile">Profile</span>
              </li>
              <li className="sidebar-item">
                <span className="e-icons info"/>
                <span className="e-text"
title="info">Info</span>
              </li>
              <li className="sidebar-item">
                <span className="e-icons settings"/>
                <span className="e-text"
title="settings">Settings</span>
              </li>
            </ul>
          </div>
        </SidebarComponent>
        <div id="main-content container-fluid col-md-12 ">
          <div className="title">Main content</div>
          <div className="sub-title"> Click the expand icon to open
and collapse icons to close the Sidebar</div>
        </div>
      </div>
    </div>
  )
}
export default App;

```

INDEX.JSX


```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));
```

See Also

- [How to add sidebar navigation](#)

Style in React Sidebar component

The following content provides the exact CSS structure that can be used to modify the component's appearance based on the user's preference.

Customizing the sidebar

Use the below CSS to customize the sidebar root element.

```
`css
.e-sidebar {
background: #898b2b
}
`
```

Customizing the sidebar based on the positions

Use the below CSS to customize the left positioned sidebar.

```
`css
.e-sidebar.e-left {
border-right: 2px solid red;
}
`
```

Use the below CSS to customize the right positioned sidebar.

```
`css
.e-sidebar.e-right {
border-left: 2px solid red;
}
`
```

Customizing the sidebar based on the active state

Use the below CSS to customize the open state of the left positioned sidebar.

```
`css
.e-sidebar.e-left.e-open {
transition: transform 2.5s ease;
}
`
```

Use the below CSS to customize the open state of the right positioned sidebar.

```
`css
.e-sidebar.e-right.e-open {
transition: transform 2.5s ease;
}
`
```

Use the below CSS to customize the closed state of the left positioned sidebar.

```
`css
.e-sidebar.e-left.e-transition.e-close {
transition: transform 2.5s ease, visibility 1200ms;
}
`
```

Use the below CSS to customize the closed state of the right positioned sidebar.

```
`css
.e-sidebar.e-right.e-transition.e-close {
transition: transform 2.5s ease, visibility 1200ms;
}
`
```

Customizing the sidebar with dock state

When you enable the Dock support, the "e-dock" class will be added to the root element. Based on that class, you can also customize all the above stated customization. Use the following CSS to customize the sidebar element with a dock state.

```
`css
.e-sidebar.e-dock {
background: #2d323e;
}
`
```

Customizing the different types of sidebar

Use the below CSS to customize the auto type sidebar.

```
`css
.e-sidebar.e-left.e-auto {
background-color: pink;
}
`
```

Use the below CSS to customize the push type sidebar.

```
`css
.e-sidebar.e-left.e-push {
background-color: beige;
}
`
```

Use the below CSS to customize the over type sidebar.

```
`css
.e-sidebar.e-left.e-over {
background-color: aqua;
}
`
```

Use the below CSS to customize the slide type sidebar.

```
`css
.e-sidebar.e-left.e-slide {
background-color: green;
}
`
```

Customizing the backdrop of the sidebar

Use the below CSS to customize the backdrop of the sidebar.

```
`css
.e-sidebar-overlay {
background-color: aqua;
}
`
```

Customizing the sidebar in the RTL direction

When you enable the RTL (right to left direction) support, the "e-rtl" class will be added to the root element. Based on that class, you can also customize all the above stated customization. Use the following CSS to customize the sidebar element in the RTL (right to left direction) mode.

```
`css
.e-sidebar.e-left.e-rtl {
background-color: antiquewhite;
}
`
```

Prevent the animation transition for the Sidebar component

Use the below CSS to prevent the animation transition for the Sidebar component.

```
`css
.e-sidebar-context .e-content-animation {
transition: none !important;
transform: none !important;
}
`
```

Accessibility in React Sidebar component

The Sidebar component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Sidebar component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

| Screen Reader Support | |

| Right-To-Left Support | |

| Color Contrast | |

| Mobile Device Support | |

| Keyboard Navigation Support | |

```
| Accessibility Checker Validation |  |
| Axe-core Accessibility Validation |  |

<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>
```

WAI-ARIA attributes

The Sidebar component followed the [WAI-ARIA](#) patterns to meet accessibility standards. By default, the Sidebar utilizes the [complementary](#) role, with the option to modify the ARIA role based on provided attributes to the root element, depending on the specific use case.

If there are multiple complementary landmark roles or aside elements in a document, it is important to provide a label for each landmark using the `aria-label` attribute. Alternatively, if the aside has a descriptive title, it can be referenced using the `aria-labelledby` attribute. This label will help assistive technology users quickly understand the purpose of each landmark.

For optimal accessibility, it is recommended to incorporate a trigger component that is navigable via a keyboard, such as a button. If this is not feasible, inclusion of the `tabIndex` attribute becomes necessary. Furthermore, explicit handling of the `aria-expanded` attribute is required for the trigger element.

Keyboard interaction

The Sidebar component does not have any inbuilt keyboard interaction support. However, you can utilize the keyboard interactions of focusable elements within the Sidebar component.

Ensuring accessibility

The Sidebar component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Sidebar component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Sidebar component with accessibility tools.

See also

- [Accessibility in Syncfusion React components](#)

How To

Sidebar with listview in React Sidebar component

Any HTML element can be placed in the Sidebar content area. Sidebar supports all types of HTML structures like [TreeView](#), [ListView](#), etc.

In the following example, the Sidebar is rendered with ListView component in its content area.

APP.JSX

```
{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { ListViewComponent } from '@syncfusion/ej2-react-lists';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';

function App() {
  let sidebarObj;
  let listObj;
  let width = '100%';
  let type = 'Over';
  let data = [
    { text: 'Home', id: 'list-01' },
    { text: 'Offers', id: 'list-02' },
    { text: 'Support', id: 'list-03' },
    { text: 'Logout', id: 'list-04' }
  ];
  function onCreate() {
    sidebarObj.element.style.visibility = '';
  }
  // Toggle (Open/Close) the Sidebar
  function toggleClick() {
    sidebarObj.toggle();
  }
  // Close the Sidebar
  function closeClick() {
    sidebarObj.hide();
  }
  return (<div className="control-section">
    { /* Initializing the Sidebar component */ }
    <SidebarComponent id="default-sidebar" ref={Sidebar => sidebarObj
= Sidebar} width={width} style={{ visibility: "hidden" }} created={onCreate}
type={type}>
      <div id='closebtn'>
        <ButtonComponent cssClass='e-normal' iconCss='e-icons e-
add-icon' onClick={toggleClick}/>
      </div>
      <div className="title">Menu</div>
      <div className='listdiv'>
        { /* Initializing the Listview inside the sidebar
component */ }
        <ListViewComponent id="listview" dataSource={data}
onClick={closeClick}/>
      </div>
      <div className="center-align">
        * ListView component is placed inside the sidebar content
area.
      </div>
    </div>
```

```

        </SidebarComponent>
        <div>
            <div className="title">Main content</div>
            <div className="sub-title"> Click the button to open/close
the Sidebar.</div>
            <div className="center-align">
                <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Toggle Sidebar</ButtonComponent>
            </div>
        </div>
    </div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { ListViewComponent } from '@syncfusion/ej2-react-lists';
import { SidebarComponent, SidebarType } from '@syncfusion/ej2-react-
navigations';
import * as React from 'react';
function App() {
    let sidebarObj: SidebarComponent;
    let listObj: SidebarComponent;
    let width: string = '100%';
    let type: SidebarType = 'Over';
    let data: Array<{ [key: string]: string }> = [
        { text: 'Home', id: 'list-01' },
        { text: 'Offers', id: 'list-02' },
        { text: 'Support', id: 'list-03' },
        { text: 'Logout', id: 'list-04' }
    ];
    function onCreate(): void {
        sidebarObj.element.style.visibility='';
    }
    // Toggle(Open/Close) the Sidebar
    function toggleClick(): void {
        sidebarObj.toggle();
    }
    // Close the Sidebar
    function closeClick(): void {
        sidebarObj.hide();
    }
    return (
        <div className="control-section">
            { /* Initializing the Sidebar component */ }
            <SidebarComponent id="default-sidebar" ref={Sidebar => sidebarObj
= Sidebar as SidebarComponent} width={width} style={{visibility:"hidden"}}
created={onCreate} type={type}>
                <div id='closebtn'>
                    <ButtonComponent cssClass='e-normal' iconCss='e-icons e-
add-icon' onClick={toggleClick}/>
                </div>
                <div className="title">Menu</div>

```

```

        <div className='listdiv'>
            /* Initializing the Listview inside the sidebar
component */
            <ListViewComponent id="listview" dataSource={data}
onClick={closeClick}/>
        </div>
        <div className="center-align">
            * ListView component is placed inside the sidebar content
area.
        </div>
    </SidebarComponent>
    <div>
        <div className="title">Main content</div>
        <div className="sub-title"> Click the button to open/close
the Sidebar.</div>
        <div className="center-align">
            <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Toggle Sidebar</ButtonComponent>
        </div>
    </div>
)
}
export default App;
{% endraw %}

```

Open close sidebar in React Sidebar component

Opening and closing the Sidebar can be achieved with built-in public methods.

- [show\(\)](#): Method to open the Sidebar.
- [hide\(\)](#): Method to close the Sidebar.
- [toggle\(\)](#): Method to toggle the open/close state of the Sidebar.

In the following sample, toggle method has been used to show or hide the Sidebar on button click.

APP.JSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
    let sidebarObj;
    function onCreate() {
        sidebarObj.element.style.visibility = '';
    }
    function open() {
        console.log("Sidebar is opened");
    }
    function close() {
        console.log("Sidebar is closed");
    }
    // Open the Sidebar
    function openClick() {
        sidebarObj.show();
    }
}

```



```

    }
    // Toggle(Open/Close) the Sidebar
    function toggleClick() {
        sidebarObj.toggle();
    }
    // Close the Sidebar
    function closeClick() {
        sidebarObj.hide();
    }
    return (<div className="control-section">
        <div id="wrapper">
            /* Initializing the Sidebar component */
            <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarObj = Sidebar} style={{ visibility: "hidden" }} close={close}
open={open} created={onCreate}>
                <div className="title"> Sidebar content</div>
                <div className="sub-title">
                    Click the button to close the Sidebar.
                </div>
                <div className="center-align">
                    <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
                </div>
            </SidebarComponent>
            <div>
                <div className="title">Main content</div>
                <div className="sub-title"> Click the button to Open the
Sidebar.</div>
                <div className="center-align">
                    <ButtonComponent onClick={openClick} id="open"
className="e-btn e-info">Open Sidebar</ButtonComponent>
                </div>
                <div className="sub-title"> Click the button to
open/close the Sidebar.</div>
                <div className="center-align">
                    <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Toggle Sidebar</ButtonComponent>
                </div>
            </div>
        </div>);
    }
    export default App;
    {% endraw %}

```

APP.TSX

```

{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
    let sidebarObj: SidebarComponent;
    function onCreate(): void {
        sidebarObj.element.style.visibility='';
    }
}

```

```

function open(): void {
    console.log("Sidebar is opened");
}
function close(): void {
    console.log("Sidebar is closed");
}
// Open the Sidebar
function openClick(): void {
    sidebarObj.show();
}
// Toggle(Open/Close) the Sidebar
function toggleClick(): void {
    sidebarObj.toggle();
}
// Close the Sidebar
function closeClick(): void {
    sidebarObj.hide();
}
return (
    <div className="control-section">
        <div id="wrapper">
            /* Initializing the Sidebar component */
            <SidebarComponent id="default-sidebar" ref={Sidebar =>
sidebarObj = Sidebar as SidebarComponent} style={{visibility:"hidden"}}
close={close} open={open} created={onCreate}>
                <div className="title"> Sidebar content</div>
                <div className="sub-title">
                    Click the button to close the Sidebar.
                </div>
                <div className="center-align">
                    <ButtonComponent onClick={closeClick} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
                </div>
            </SidebarComponent>
            <div>
                <div className="title">Main content</div>
                <div className="sub-title"> Click the button to Open the
Sidebar.</div>
                <div className="center-align">
                    <ButtonComponent onClick={openClick} id="open"
className="e-btn e-info">Open Sidebar</ButtonComponent>
                </div>
                <div className="sub-title"> Click the button to
open/close the Sidebar.</div>
                <div className="center-align">
                    <ButtonComponent onClick={toggleClick} id="toggle"
className="e-btn e-info">Toggle Sidebar</ButtonComponent>
                </div>
            </div>
        </div>
    </div>
)
}
export default App;
{% endraw %}

```

Multiple sidebar in React Sidebar component

Two Sidebars can be initialized in a web page with same main content. Sidebars can be initialized on right side or left side of the main content using [position](#) property.

The HTML element with class name `e-main-content` will be considered as the main content and both the

Sidebars will behave as side content to this main content area.

APP.JSX

```
{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import * as React from 'react';
function App() {
  let leftSidebarObj;
  let rightSidebarObj;
  let type = 'Push';
  function onCreate() {
    leftSidebarObj.element.style.visibility = '';
    rightSidebarObj.element.style.visibility = '';
  }
  // Toggle(Open/Close) the Sidebar1
  function leftToggle() {
    leftSidebarObj.toggle();
  }
  // Toggle(Open/Close) the Sidebar2
  function rightToggle() {
    rightSidebarObj.toggle();
  }
  return (<div className="control-section">
    <div id="wrapper">
      /* Initializing the Sidebar component */
      <SidebarComponent id="default" ref={Sidebar => leftSidebarObj
= Sidebar} width="200px" type={type} created={onCreate} style={{ visibility:
"hidden" }}>
        <div className="title"> Left Sidebar content</div>
      </SidebarComponent>
      /* Initializing the another Sidebar component */
      <SidebarComponent id="default1" ref={Sidebar =>
rightSidebarObj = Sidebar} width="200px" type={type} position="Right"
created={onCreate} style={{ visibility: "hidden" }}>
        <div className="title"> Right Sidebar content</div>
      </SidebarComponent>
      <div className="e-main-content">
        <p>Place your main content here.....</p>
        <ButtonComponent onClick={leftToggle} id="toggle-btn"
className="e-btn e-info">Toggle Sidebar1</ButtonComponent>
        <br /><br />
        <ButtonComponent onClick={rightToggle} id="toggle-btn1"
className="e-btn e-info">Toggle Sidebar2</ButtonComponent>
      </div>
    </div>
  </div>);
}
export default App;
```

```
{% endraw %}
```

APP.TSX

```
{% raw %}
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent, SidebarType } from '@syncfusion/ej2-react-
navigations';
import * as React from 'react';
function App() {
    let leftSidebarObj: SidebarComponent;
    let rightSidebarObj: SidebarComponent;
    let type: SidebarType = 'Push'
    function onCreate(): void {
        leftSidebarObj.element.style.visibility='';
        rightSidebarObj.element.style.visibility='';
    }
    // Toggle(Open/Close) the Sidebar1
    function leftToggle(): void {
        leftSidebarObj.toggle();
    }
    // Toggle(Open/Close) the Sidebar2
    function rightToggle(): void {
        rightSidebarObj.toggle();
    }
    return (
        <div className="control-section">
            <div id="wrapper">
                /* Initializing the Sidebar component */
                <SidebarComponent id="default" ref={Sidebar => leftSidebarObj
= Sidebar as SidebarComponent}
                    width="200px" type={type} created={onCreate}
style={{visibility:"hidden"}}>
                    <div className="title"> Left Sidebar content</div>
                </SidebarComponent>
                /* Initializing the another Sidebar component */
                <SidebarComponent id="default1" ref={Sidebar =>
rightSidebarObj = Sidebar as SidebarComponent}
                    width="200px" type={type} position="Right"
created={onCreate} style={{visibility:"hidden"}}>
                    <div className="title"> Right Sidebar content</div>
                </SidebarComponent>
                <div className="e-main-content">
                    <p>Place your main content here.....</p>
                    <ButtonComponent onClick={leftToggle} id="toggle-btn"
className="e-btn e-info">Toggle Sidebar1</ButtonComponent>
                    <br/><br/>
                    <ButtonComponent onClick={rightToggle} id="toggle-btn1"
className="e-btn e-info">Toggle Sidebar2</ButtonComponent>
                </div>
            </div>
        </div>
    )
}
export default App;
{% endraw %}
```

Sidebar with treeview in React Sidebar component

The following example demonstrates how to render TreeView component inside the Sidebar with dock state and how to achieve expand and collapse the functionalities simultaneously in the sidebar and Treeview.

On collapse, the LI elements of TreeView show icons only to represent the short sign of the hidden text content. On expand, hidden text content will be set to be visible.

APP.JSX

```
{% raw %}
import * as React from 'react';
import { SidebarComponent, TreeViewComponent } from '@syncfusion/ej2-react-navigations';
function App() {
    let sidebarobj;
    let treeviewobj;
    let data = [
        {
            nodeId: '01', nodeText: 'Installation', iconCss: 'icon-microchip icon',
            nodeChild: [
                { nodeId: '05-01', nodeText: 'Calendar', iconCss: 'icon-circle-thin icon' },
                { nodeId: '05-02', nodeText: 'DatePicker', iconCss: 'icon-circle-thin icon' },
                { nodeId: '05-03', nodeText: 'DateTimePicker', iconCss: 'icon-circle-thin icon' },
                { nodeId: '05-04', nodeText: 'DateRangePicker', iconCss: 'icon-circle-thin icon' },
                { nodeId: '05-05', nodeText: 'TimePicker', iconCss: 'icon-circle-thin icon' },
                { nodeId: '05-06', nodeText: 'SideBar', iconCss: 'icon-circle-thin icon' }
            ]
        },
        {
            nodeId: '02', nodeText: 'Deployment', iconCss: 'icon-thumbs-up-alt icon',
        },
        {
            nodeId: '03', nodeText: 'Quick Start', iconCss: 'icon-docs icon',
        },
        {
            nodeId: '04', nodeText: 'Components', iconCss: 'icon-th icon',
            nodeChild: [
                { nodeId: '04-01', nodeText: 'Calendar', iconCss: 'icon-circle-thin icon' },
                { nodeId: '04-02', nodeText: 'DatePicker', iconCss: 'icon-circle-thin icon' },
                { nodeId: '04-03', nodeText: 'DateTimePicker', iconCss: 'icon-circle-thin icon' },
                { nodeId: '04-04', nodeText: 'DateRangePicker', iconCss: 'icon-circle-thin icon' },
            ]
        }
    ]
}
```

```

        { nodeId: '04-05', nodeText: 'TimePicker', iconCss: 'icon-
circle-thin icon' },
        { nodeId: '04-06', nodeText: 'SideBar', iconCss: 'icon-
circle-thin icon' }
    ]
},
{
    nodeId: '05', nodeText: 'API Reference', iconCss: 'icon-code
icon',
    nodeChild: [
        { nodeId: '05-01', nodeText: 'Calendar', iconCss: 'icon-
circle-thin icon' },
        { nodeId: '05-02', nodeText: 'DatePicker', iconCss: 'icon-
circle-thin icon' },
        { nodeId: '05-03', nodeText: 'DateTimePicker', iconCss:
'icon-circle-thin icon' },
        { nodeId: '05-04', nodeText: 'DateRangePicker', iconCss:
'icon-circle-thin icon' },
        { nodeId: '05-05', nodeText: 'TimePicker', iconCss: 'icon-
circle-thin icon' },
        { nodeId: '05-06', nodeText: 'SideBar', iconCss: 'icon-
circle-thin icon' }
    ]
},
{
    nodeId: '06', nodeText: 'Browser Compatibility', iconCss: 'icon-
chrome icon'
},
{
    nodeId: '07', nodeText: 'Upgrade Packages', iconCss: 'icon-up-
hand icon'
},
{
    nodeId: '08', nodeText: 'Release Notes', iconCss: 'icon-bookmark-
empty icon'
},
{
    nodeId: '09', nodeText: 'FAQ', iconCss: 'icon-help-circled icon'
},
{
    nodeId: '10', nodeText: 'License', iconCss: 'icon-doc-text icon'
}
];
let width = '290px';
let target = '.main-content';
let mediaQuery = '(min-width: 600px)';
let dockSize = "44px";
let fields = { dataSource: data, id: 'nodeId', text: 'nodeText', child:
'nodeChild' };
function onCreate() {
    sidebarobj.element.style.visibility = '';
}
function onClose() {
    treeviewobj.collapseAll();
}
function toggleClick() {
    if (sidebarobj.isOpen) {

```

```

        sidebarobj.hide();
        treeviewobj.collapseAll();
    }
    else {
        sidebarobj.show();
        treeviewobj.expandAll();
    }
}
let sidbarToggle = null;
return (
    // Sidebar Element Declaration
    <div className="control-section">
        <div id="wrapper">
            <title>Essential JS 2 for React - Sidebar {'>'} Sidebar with
ListView </title>
            <div className="col-lg-12 col-sm-12 col-md-12">
                <div className='main-header' id='header-section'>
                    <ul className='header-list'>
                        <li className='float-left header-style icon-menu'
id='hamburger' onClick={toggleClick} ref={sidebar => sidbarToggle =
sidebar}></li>
                        <li className='float-left header-style nav-
pane'><b>Navigation Pane</b></li>
                        <li className='header-style float-right support
border-left'><b>Support</b></li>
                    </ul>
                </div>
                <SidebarComponent id="sidebar-treeview" ref={Sidebar =>
sidebarobj = Sidebar} width={width} target={target} mediaQuery={mediaQuery}
style={{ visibility: "hidden" }} created={onCreate} close={onClose}
dockSize={dockSize} enableDock={true}>
                    <div className='main-menu'>
                        <div>
                            <TreeViewComponent id='main-treeview'
ref={TreeView => treeviewobj = Treeview} fields={fields} expandOn='Click' />
                        </div>
                    </div>
                </SidebarComponent>
                <div className="main-content" id="main-text">
                    <div className='sidebar-content'>
                        <h2 className='sidebar-heading'> Responsive
Sidebar With Treeview</h2>
                        <p className='paragraph-content'> This is a
graphical aid for visualising and categorising the site,
                        in the style of an expandable and collapsable
treeview component. It auto-expands to display the node(s),
                        if any, corresponding to the currently viewed
title, highlighting that node(s) and its ancestors.
                        Load-on-demand when expanding nodes is
available where supported (most graphical browsers), falling
                        back to a full-page reload. MediaWiki-
supported caching, aside from squid, has been considered so
                        that unnecessary re-downloads of content are
avoided where possible. The complete expanded/collapsed
                        state of the treeview persists across page
views in most situations.</p>
                    </div>
                </div>
            </div>
        </div>
    </div>
)

```

```

        <p className='paragraph-content'>Lorem ipsum
dolor sit amet, consectetur adipisicing elit, sed do
        eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis
        nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure
        dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur.
        Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim
        id est laborum.</p>
        <div className='line'></div>
        <h2 className='sidebar-heading'>Lorem Ipsum
Dolor</h2>
        <p className='paragraph-content'>Lorem ipsum
dolor sit amet, consectetur adipisicing elit, sed do
        eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis
        nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure
        dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur.</p>
        <div className='line'></div>
        <h2 className='sidebar-heading'> Lorem Ipsum
Dolor</h2>
        <p className='paragraph-content'>Lorem ipsum
dolor sit amet, consectetur adipisicing elit, sed do eiusmod
        tempor incididunt ut labore et dolore
magna aliqua. Ut enim ad minim veniam, quis nostrud
        exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in
        reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint
        occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.</p>
        <div className='line'></div>
        <h2 className='sidebar-heading'> Lorem Ipsum
Dolor</h2>
        <p className='paragraph-content'>Lorem ipsum
dolor sit amet, consectetur adipisicing elit, sed do eiusmod
        tempor incididunt ut labore et dolore
magna aliqua. Ut enim ad minim veniam, quis nostrud
        exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in
        reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint
        occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.</p>
        </div>
    </div>
</div>
</div>
</div>);
}
export default App;
{% endraw %}

```


APP.TSX

```

{% raw %}
import * as React from 'react';
import { SidebarComponent, TreeViewComponent, TreeView } from
 '@syncfusion/ej2-react-navigations';
import { enableRipple } from '@syncfusion/ej2-base';
function App() {
    let sidebarobj: SidebarComponent;
    let treeviewobj: TreeViewComponent;
    let data: any = [
        {
            nodeId: '01', nodeText: 'Installation', iconCss: 'icon-microchip
            icon',
            nodeChild: [
                { nodeId: '05-01', nodeText: 'Calendar', iconCss: 'icon-
                circle-thin icon' },
                { nodeId: '05-02', nodeText: 'DatePicker', iconCss: 'icon-
                circle-thin icon' },
                { nodeId: '05-03', nodeText: 'DateTimePicker', iconCss:
                'icon-circle-thin icon' },
                { nodeId: '05-04', nodeText: 'DateRangePicker', iconCss:
                'icon-circle-thin icon' },
                { nodeId: '05-05', nodeText: 'TimePicker', iconCss: 'icon-
                circle-thin icon' },
                { nodeId: '05-06', nodeText: 'SideBar', iconCss: 'icon-
                circle-thin icon' }
            ]
        },
        {
            nodeId: '02', nodeText: 'Deployment', iconCss: 'icon-thumbs-up-
            alt icon',
        },
        {
            nodeId: '03', nodeText: 'Quick Start', iconCss: 'icon-docs icon',
        },
        {
            nodeId: '04', nodeText: 'Components', iconCss: 'icon-th icon',
            nodeChild: [
                { nodeId: '04-01', nodeText: 'Calendar', iconCss: 'icon-
                circle-thin icon' },
                { nodeId: '04-02', nodeText: 'DatePicker', iconCss: 'icon-
                circle-thin icon' },
                { nodeId: '04-03', nodeText: 'DateTimePicker', iconCss:
                'icon-circle-thin icon' },
                { nodeId: '04-04', nodeText: 'DateRangePicker', iconCss:
                'icon-circle-thin icon' },
                { nodeId: '04-05', nodeText: 'TimePicker', iconCss: 'icon-
                circle-thin icon' },
                { nodeId: '04-06', nodeText: 'SideBar', iconCss: 'icon-
                circle-thin icon' }
            ]
        },
        {

```

```

        nodeId: '05', nodeText: 'API Reference', iconCss: 'icon-code
icon',
        nodeChild: [
            { nodeId: '05-01', nodeText: 'Calendar', iconCss: 'icon-
circle-thin icon' },
            { nodeId: '05-02', nodeText: 'DatePicker', iconCss: 'icon-
circle-thin icon' },
            { nodeId: '05-03', nodeText: 'DateTimePicker', iconCss:
'icon-circle-thin icon' },
            { nodeId: '05-04', nodeText: 'DateRangePicker', iconCss:
'icon-circle-thin icon' },
            { nodeId: '05-05', nodeText: 'TimePicker', iconCss: 'icon-
circle-thin icon' },
            { nodeId: '05-06', nodeText: 'SideBar', iconCss: 'icon-
circle-thin icon' }
        ]
    },
    {
        nodeId: '06', nodeText: 'Browser Compatibility', iconCss: 'icon-
chrome icon'
    },
    {
        nodeId: '07', nodeText: 'Upgrade Packages', iconCss: 'icon-up-
hand icon'
    },
    {
        nodeId: '08', nodeText: 'Release Notes', iconCss: 'icon-bookmark-
empty icon'
    },
    {
        nodeId: '09', nodeText: 'FAQ', iconCss: 'icon-help-circled icon'
    },
    {
        nodeId: '10', nodeText: 'License', iconCss: 'icon-doc-text icon'
    }
];
let width: string = '290px';
let target: string = '.main-content';
let mediaQuery: string = '(min-width: 600px)';
let dockSize: string = "44px";
let fields: object = { dataSource: data, id: 'nodeId', text: 'nodeText',
child: 'nodeChild' };
function onCreate(): void {
    sidebarobj.element.style.visibility = '';
}
function onClose(): void {
    treeviewobj.collapseAll();
}
function toggleClick():void {
    if(sidebarobj.isOpen)
    {
        sidebarobj.hide();
        treeviewobj.collapseAll();
    }
    else {
        sidebarobj.show();
        treeviewobj.expandAll();
    }
}

```

```

    }
  }
  let sidebarToggle = null;
  return (
    // Sidebar Element Declaration
    <div className="control-section">
      <div id="wrapper">
        <title>Essential JS 2 for React - Sidebar {'>'} Sidebar with
        ListView </title>
        <div className="col-lg-12 col-sm-12 col-md-12">
          <div className='main-header' id='header-section'>
            <ul className='header-list'>
              <li className='float-left header-style icon-menu'
id='hamburger' onClick={toggleClick} ref={sidebar=>(sidebarToggle as
any)=sidebar}></li>
              <li className='float-left header-style nav-
pane'><b>Navigation Pane</b></li>
              <li className='header-style float-right support
border-left'><b>Support</b></li>
            </ul>
          </div>
          <SidebarComponent id="sidebar-treeview" ref={Sidebar =>
sidebarobj = Sidebar as SidebarComponent} width={width} target={target}
mediaQuery={mediaQuery} style={{visibility:"hidden"}} created={onCreate}
close={onClose} dockSize={dockSize} enableDock={true}>
            <div className='main-menu'>
              <div>
                <TreeViewComponent id='main-treeview'
ref={TreeView => treeviewobj = Treeview as TreeViewComponent}fields={fields}
expandOn='Click' />
              </div>
            </div>
          </SidebarComponent>
          <div className="main-content" id="main-text">
            <div className='sidebar-content'>
              <h2 className='sidebar-heading'> Responsive
Sidebar With Treeview</h2>
              <p className='paragraph-content'> This is a
graphical aid for visualising and categorising the site,
in the style of an expandable and collapsable
treeview component. It auto-expands to display the node(s),
if any, corresponding to the currently viewed
title, highlighting that node(s) and its ancestors.
Load-on-demand when expanding nodes is
available where supported (most graphical browsers), falling
back to a full-page reload. MediaWiki-
supported caching, aside from squid, has been considered so
that unnecessary re-downloads of content are
avoided where possible. The complete expanded/collapsed
state of the treeview persists across page
views in most situations.</p>
              <p className='paragraph-content'>Lorem ipsum
dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure

```

```

        dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur.
        Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim
        id est laborum.</p>
        <div className='line'></div>
        <h2 className='sidebar-heading'>Lorem Ipsum
Dolor</h2>
        <p className='paragraph-content'>Lorem ipsum
dolor sit amet, consectetur adipisicing elit, sed do
        eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis
        nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure
        dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur.</p>
        <div className='line'></div>
        <h2 className='sidebar-heading'> Lorem Ipsum
Dolor</h2>
        <p className='paragraph-content'>Lorem ipsum
dolor sit amet, consectetur adipisicing elit, sed do eiusmod
        tempor incididunt ut labore et dolore
magna aliqua. Ut enim ad minim veniam, quis nostrud
        exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in
        reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint
        occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.</p>
        <div className='line'></div>
        <h2 className='sidebar-heading'> Lorem Ipsum
Dolor</h2>
        <p className='paragraph-content'>Lorem ipsum
dolor sit amet, consectetur adipisicing elit, sed do eiusmod
        tempor incididunt ut labore et dolore
magna aliqua. Ut enim ad minim veniam, quis nostrud
        exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in
        reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint
        occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.</p>
    </div>
  </div>
</div>
)
}
export default App;
{% endraw %}

```

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';

```

```
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));
```

Fixed sidebar in React Sidebar component

The Sidebar does not require any specific style to make it as a fixed one. By default, the Sidebar position will be in fixed state. The following example demonstrates that the Sidebar is rendered with a fixed position. The position of the Sidebar will not change when scrolling the main content area.

APP.JSX

```
{% raw %}
import * as React from 'react';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
function App() {
  let sidebarobj;
  let width = '260px';
  function toggleClick() {
    sidebarobj.toggle();
  }
  function onCreate() {
    sidebarobj.element.style.visibility = '';
  }
  return (
    // Sidebar Element Declaration
    <div className="control-section">
      <SidebarComponent id="default-sidebar" ref={Sidebar => sidebarobj
= Sidebar} style={{ visibility: "hidden" }} width={width} created={onCreate}>
        <div className="sidebar-header header-cover" style={{
backgroundColor: '#0378d5' }}>
          <div className="image-container">
            <div className="sidebar-image">
              </div>
            </div>
            <div style={{ padding: '0 0 5px 0' }}>
              <a className="sidebar-brand" href="#settings-
dropdown">
                john.doe@gmail.com
              </a>
              <span className="sf-icon-down icon"></span>
            </div>
          </div>
          <ul className="nav sidebar-nav">
            <li>
              <a href="#">
                <i className="sf-icon-sidebar sf-icon-file"></i>
                <span className="e-text"> Inbox</span>
              </a>
            </li>
          </ul>
        </div>
      </div>
    </div>
  );
}
```

```

        <li>
            <a href="#">
                <i className="sf-icon-sidebar sf-icon-
starred"></i>
                <span className="e-text"> Starred</span>
            </a>
        </li>
        <li>
            <a href="#">
                <i className="sf-icon-sidebar sf-icon-
recent"></i>
                <span className="e-text"> Snoozed</span>
            </a>
        </li>
        <li>
            <a href="#">
                <i className="sf-icon-sidebar sf-icon-
important"></i>
                <span className="e-text"> Important</span>
            </a>
        </li>
        <li>
            <a href="#">
                <i className="sf-icon-sidebar sf-icon-
offline"></i>
                <span className="e-text"> Sent</span>
            </a>
        </li>
        <li>
            <a href="#">
                <i className="sf-icon-sidebar sf-icon-
backup"></i>
                <span className="e-text"> Draft</span>
            </a>
        </li>
    </ul>
</SidebarComponent>
{ /* End of Sidebar Declaration */ }
{ /* Main Content Declaration */ }
<div>
    <div className="content">
        <div id="left">
            <span id="hamburger" onClick={toggleClick}
className="e-icons menu default"></span>
        </div>
        <div id="center">
            <span>Inbox</span>
        </div>
        <div id="right">
            <span className="sf-icon-search"></span>
        </div>
    </div>
    <div>
        <div className="e-control e-listview e-list-template e-
touch">
            <ul className="e-list-parent e-ul ">

```

```

<li className="e-list-group-item e-level-1"
role="group" data-uid="group-list-item-Today">
  <div className="e-text-content"
role="presentation"><span style={{ width: '100%', marginLeft: '2%',
marginTop: '-2%' }}>Today</span></div>
</li>
<li className="e-list-item">
  <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
    <span className="e-avatar e-icon sf-icon-
profile"></span>
    <div>
      <span className="e-list-item-header
e-name">Albert Lives</span>
      </div>
      <span className="received e-list-content
e-second-heading">Opening for Sales Manager</span>
      <span className="e-list-item-header sf-
icon-star">
        <span className="e-list-text">Hello
Uta Morgan,</span></span>
      </div>
    </li>
    <li className="e-list-item">
      <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
        <span className="e-avatar e-icon sf-icon-
profile"></span>
        <div>
          <span className="e-list-item-header
e-name">
            Ila Russo</span>
          </div>
          <span className="received e-list-content
e-second-heading">Business dinner invitation
          </span>
          <span className="e-list-item-header sf-
icon-star">
            <span className="e-list-text">Hello
Jelani Moreno,</span></span>
          </div>
        </li>
        <li className="e-list-item">
          <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
            <span className="e-avatar e-icon sf-icon-
profile"></span>
            <div>
              <span className="e-list-item-header
e-name">
                Garth Owen</span>
              </div>
              <span className="received e-list-content
e-second-heading">Application for Job Title</span>
              <span className="e-list-item-header sf-
icon-star">

```

```

Ila Russo,</span></span>
                                <span className="e-list-text">Hello
                                </div>
                                </li>
                                <li className="e-list-item">
                                <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
                                <span className="e-avatar e-icon sf-icon-
profile"></span>
                                <div>
                                <span className="e-list-item-header
e-name">Ursula Patterson</span>
                                </div>
                                <span className="received e-list-content
e-second-heading">Hello Kerry Best,</span>
                                <span className="e-list-item-header sf-
icon-star">
                                <span className="e-list-
text">Programmer Position Application</span></span>
                                </div>
                                </li>
                                <li className="e-list-item">
                                <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
                                <span className="e-avatar e-icon sf-icon-
profile"></span>
                                <div>
                                <span className="e-list-item-header
e-name">
                                Nichole Rivas</span>
                                </div>
                                <span className="received e-list-content
e-second-heading">Annual Conference</span>
                                <span className="e-list-item-header sf-
icon-star">
                                <span className="e-list-text">Hi Igor
Mccoy,</span></span>
                                </div>
                                </li>
                                <li>
                                <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
                                <span className="e-avatar e-icon sf-icon-
profile"></span>
                                <div>
                                <span className="e-list-item-header
e-name">
                                Nichole Rivas</span>
                                </div>
                                <span className="received e-list-content
e-second-heading">Annual Conference</span>
                                <span className="e-list-item-header sf-
icon-star">
                                <span className="e-list-text">Hi Igor
Mccoy,</span></span>
                                </div>
                                </li>

```



```

<li className="e-list-item">
  <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
    <span className="e-avatar e-icon sf-icon-
profile"></span>
    <div>
      <span className="e-list-item-header
e-name">
        Nichole Rivas</span>
      </div>
      <span className="received e-list-content
e-second-heading">Annual Conference</span>
      <span className="e-list-item-header sf-
icon-star">
        <span className="e-list-text">Hi Igor
Mccoy,</span></span>
      </div>
    </li>
    <li className="e-list-item">
      <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
        <span className="e-avatar e-icon sf-icon-
profile"></span>
        <div>
          <span className="e-list-item-header
e-name">
            Nichole Rivas</span>
          </div>
          <span className="received e-list-content
e-second-heading">Annual Conference</span>
          <span className="e-list-item-header sf-
icon-star">
            <span className="e-list-text">Hi Igor
Mccoy,</span></span>
          </div>
        </li>
        <li className="e-list-item">
          <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
            <span className="e-avatar e-icon sf-icon-
profile"></span>
            <div>
              <span className="e-list-item-header
e-name">
                Nichole Rivas</span>
              </div>
              <span className="received e-list-content
e-second-heading">Annual Conference</span>
              <span className="e-list-item-header sf-
icon-star">
                <span className="e-list-text">Hi Igor
Mccoy,</span></span>
              </div>
            </li>
            <li className="e-list-item">
              <div className="e-list-wrapper e-list-avatar
e-list-multi-line">

```

```

                                <span className="e-avatar e-icon sf-icon-
profile"></span>
                                <div>
                                    <span className="e-list-item-header
e-name">Ursula Patterson</span>
                                </div>
                                <span className="received e-list-content
e-second-heading">Hello Kerry Best,</span>
                                <span className="e-list-item-header sf-
icon-star">
                                    <span className="e-list-
text">Programmer Position Application</span></span>
                                </div>
                            </li>
                        </ul>
                    </div>
                </div>
            </div>);
        }
        export default App;
        {% endraw %}

```

APP.TSX

```

{% raw %}
import * as React from 'react';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
import { enableRipple } from '@syncfusion/ej2-base';
function App() {
    let sidebarobj: SidebarComponent;
    let width: string = '260px';
    function toggleClick():void {
        sidebarobj.toggle();
    }
    function onCreate(): void {
        sidebarobj.element.style.visibility = '';
    }
    return (
        // Sidebar Element Declaration
        <div className="control-section">
            <SidebarComponent id="default-sidebar" ref={Sidebar => sidebarobj
= Sidebar as SidebarComponent} style={{visibility:"hidden"}} width={width}
created={onCreate}>
                <div className="sidebar-header header-cover"
style={{backgroundColor: '#0378d5'}}>
                    <div className="image-container">
                        <div className="sidebar-image">
                            </div>
                        </div>
                        <div style={{padding: '0 0 5px 0'}}>
                            <a className="sidebar-brand" href="#settings-
dropdown">
                                john.doe@gmail.com
                            </a>
                            <span className="sf-icon-down icon"></span>

```

```

        </div>
    </div>
    <ul className="nav sidebar-nav">
        <li>
            <a href="#">
                <i className="sf-icon-sidebar sf-icon-file"></i>
                <span className="e-text"> Inbox</span>
            </a>
        </li>
        <li>
            <a href="#">
                <i className="sf-icon-sidebar sf-icon-
starred"></i>
                <span className="e-text"> Starred</span>
            </a>
        </li>
        <li>
            <a href="#">
                <i className="sf-icon-sidebar sf-icon-
recent"></i>
                <span className="e-text"> Snoozed</span>
            </a>
        </li>
        <li>
            <a href="#">
                <i className="sf-icon-sidebar sf-icon-
important"></i>
                <span className="e-text"> Important</span>
            </a>
        </li>
        <li>
            <a href="#">
                <i className="sf-icon-sidebar sf-icon-
offline"></i>
                <span className="e-text"> Sent</span>
            </a>
        </li>
        <li>
            <a href="#">
                <i className="sf-icon-sidebar sf-icon-
backup"></i>
                <span className="e-text"> Draft</span>
            </a>
        </li>
    </ul>
</SidebarComponent>
{ /* End of Sidebar Declaration */}
{ /* Main Content Declaration */}
<div>
    <div className="content">
        <div id="left">
            <span id="hamburger"
onClick={toggleClick}className="e-icons menu default"></span>
        </div>
        <div id="center">
            <span>Inbox</span>
        </div>
    </div>

```

```

        <div id="right">
            <span className="sf-icon-search"></span>
        </div>
    </div>
    <div>
        <div className="e-control e-listview e-list-template e-
touch">
            <ul className="e-list-parent e-ul ">
                <li className="e-list-group-item e-level-1"
role="group" data-uid="group-list-item-Today"
>
                    <div className="e-text-content"
role="presentation"><span style={{width: '100%', marginLeft: '2%',
marginTop: '-2%'}}>Today</span></div>
                </li>
                <li className="e-list-item">
                    <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
                        <span className="e-avatar e-icon sf-icon-
profile"></span>
                        <div>
                            <span className="e-list-item-header
e-name">Albert Lives</span>
                            </div>
                            <span className="received e-list-content
e-second-heading">Opening for Sales Manager</span>
                            <span className="e-list-item-header sf-
icon-star">
                                <span className="e-list-text">Hello
Uta Morgan,</span></span>
                            </div>
                        </li>
                        <li className="e-list-item">
                            <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
                                <span className="e-avatar e-icon sf-icon-
profile"></span>
                                <div>
                                    <span className="e-list-item-header
e-name">
                                        Ila Russo</span>
                                    </div>
                                    <span className="received e-list-content
e-second-heading">Business dinner invitation
                                    </span>
                                    <span className="e-list-item-header sf-
icon-star">
                                        <span className="e-list-text">Hello
Jelani Moreno,</span></span>
                                    </div>
                                </li>
                                <li className="e-list-item">
                                    <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
                                        <span className="e-avatar e-icon sf-icon-
profile"></span>
                                        <div>

```

```

                                <span className="e-list-item-header
e-name">
                                Garth Owen</span>
                                </div>
                                <span className="received e-list-content
e-second-heading">Application for Job Title</span>
                                <span className="e-list-item-header sf-
icon-star">
                                <span className="e-list-text">Hello
Ila Russo,</span></span>
                                </div>
                                </li>
                                <li className="e-list-item">
                                <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
                                <span className="e-avatar e-icon sf-icon-
profile"></span>
                                <div>
                                <span className="e-list-item-header
e-name">Ursula Patterson</span>
                                </div>
                                <span className="received e-list-content
e-second-heading">Hello Kerry Best,</span>
                                <span className="e-list-item-header sf-
icon-star">
                                <span className="e-list-
text">Programmer Position Application</span></span>
                                </div>
                                </li>
                                <li className="e-list-item">
                                <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
                                <span className="e-avatar e-icon sf-icon-
profile"></span>
                                <div>
                                <span className="e-list-item-header
e-name">
                                Nichole Rivas</span>
                                </div>
                                <span className="received e-list-content
e-second-heading">Annual Conference</span>
                                <span className="e-list-item-header sf-
icon-star">
                                <span className="e-list-text">Hi Igor
Mccoy,</span></span>
                                </div>
                                </li>
                                <li>
                                <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
                                <span className="e-avatar e-icon sf-icon-
profile"></span>
                                <div>
                                <span className="e-list-item-header
e-name">
                                Nichole Rivas</span>
                                </div>

```

```

<span className="received e-list-content
e-second-heading">Annual Conference</span>
<span className="e-list-item-header sf-
icon-star">
<span className="e-list-text">Hi Igor
Mccoy,</span></span>
</div>
</li>
<li className="e-list-item">
<div className="e-list-wrapper e-list-avatar
e-list-multi-line">
<span className="e-avatar e-icon sf-icon-
profile"></span>
<div>
<span className="e-list-item-header
e-name">
<span>Nichole Rivas</span>
</div>
<span className="received e-list-content
e-second-heading">Annual Conference</span>
<span className="e-list-item-header sf-
icon-star">
<span className="e-list-text">Hi Igor
Mccoy,</span></span>
</div>
</li>
<li className="e-list-item">
<div className="e-list-wrapper e-list-avatar
e-list-multi-line">
<span className="e-avatar e-icon sf-icon-
profile"></span>
<div>
<span className="e-list-item-header
e-name">
<span>Nichole Rivas</span>
</div>
<span className="received e-list-content
e-second-heading">Annual Conference</span>
<span className="e-list-item-header sf-
icon-star">
<span className="e-list-text">Hi Igor
Mccoy,</span></span>
</div>
</li>
<li className="e-list-item">
<div className="e-list-wrapper e-list-avatar
e-list-multi-line">
<span className="e-avatar e-icon sf-icon-
profile"></span>
<div>
<span className="e-list-item-header
e-name">
<span>Nichole Rivas</span>
</div>
<span className="received e-list-content
e-second-heading">Annual Conference</span>
```

```

                                <span className="e-list-item-header sf-
icon-star">
                                <span className="e-list-text">Hi Igor
Mccoy,</span></span>
                                </div>
                                </li>
                                <li className="e-list-item">
                                <div className="e-list-wrapper e-list-avatar
e-list-multi-line">
                                <span className="e-avatar e-icon sf-icon-
profile"></span>
                                <div>
                                <span className="e-list-item-header
e-name">Ursula Patterson</span>
                                </div>
                                <span className="received e-list-content
e-second-heading">Hello Kerry Best,</span>
                                <span className="e-list-item-header sf-
icon-star">
                                <span className="e-list-
text">Programmer Position Application</span></span>
                                </div>
                                </li>
                                </ul>
                                </div>
                                </div>
                                </div>
                                </div>
                                </div>
                                )
    }
    export default App;
    {% enddraw %}

```

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

Sidebar with variation animation in React Sidebar component

In the following example, the sidebar is rendered with custom animation effects. Click the buttons available in the main content area to check how the custom animations works with sidebar.

Sidebar will automatically adjust expanding animation to match any custom size specified in **CSS** styles.

APP.JSX

```

{% raw %}
import * as React from 'react';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent } from '@syncfusion/ej2-react-navigations';
function App() {
    let sidebarObj;
    let btnObj;
    let showBackdrop = true;
    function onCreate() {
        sidebarObj.element.style.visibility = '';
    }
    // Zoom Sidebar
    function zoom() {
        sidebarObj.show();
        sidebarObj.element.classList.add("w3-animate-zoom");
    }
    // Open Door
    function open_door() {
        sidebarObj.show();
        const sidebar = document.getElementsByClassName("e-sidebar-
overlay")[0];
        sidebar.classList.add("move");
    }
    // Bottom to Top
    function bottom_top() {
        sidebarObj.show();
        sidebarObj.element.classList.add("w3-animate-bottom");
    }
    // Rotate
    function rotate() {
        sidebarObj.show();
        sidebarObj.element.classList.add("rotate");
    }
    // Rotate 3D Effect
    function rotate_3d() {
        sidebarObj.show();
        sidebarObj.element.classList.add("rotate_3d");
    }
    // Reverse Slide Out
    function reverse() {
        sidebarObj.show();
        sidebarObj.element.classList.add("reverse_slide_out");
    }
    // Close Sidebar Element
    function close() {
        sidebarObj.element.classList.remove("sidebar");
        sidebarObj.element.classList.remove("rotate");
        sidebarObj.element.classList.remove("w3-animate-zoom");
        sidebarObj.element.classList.remove("w3-animate-bottom");
        sidebarObj.element.classList.remove("rotate_3d");
        sidebarObj.element.classList.remove("reverse_slide_out");
        sidebarObj.hide();
    }
    return (
        // Declaration of Sidebar Element
        <div className="control-section">

```



```

        <SidebarComponent id="sidebar-element" className="sidebar"
ref={Sidebar => sidebarObj = Sidebar} showBackdrop={showBackdrop} style={{
visibility: "hidden" }} created={onCreate} width="280px">
        <div className="title"> Sidebar content</div>
        <div className="sub-title">
            * Sidebar is rendered with animation effect
        </div>
        <div className="center-align">
            { /* Close Button Declaration */ }
            <ButtonComponent onClick={close} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
        </div>
    </SidebarComponent>
    { /* End of Sidebar Element Declaration */ }
    { /* Main Content Declaration */ }
    <div className="content">
        <div className="title">Sidebar Transitions</div>
        <div className="sub-title"> * Click the below button to
render the Sidebar with animation effect.</div>
        <div className="center-align">
            { /* Button Declaaration */ }
            <ButtonComponent onClick={zoom} id="zoom" className="e-
btn e-info">Zoom Sidebar</ButtonComponent>
            <ButtonComponent onClick={open_door} id="open_door"
className="e-btn e-info">Open Door</ButtonComponent>
            <ButtonComponent onClick={bottom_top} id="bottom_top"
className="e-btn e-info">Bottom to Top </ButtonComponent>
        </div>
        <div className="center-align">
            <ButtonComponent onClick={rotate} id="rotate"
className="e-btn e-info">Rotate</ButtonComponent>
            <ButtonComponent onClick={rotate_3d} id="rotate_3d"
className="e-btn e-info">Rotate 3D</ButtonComponent>
            <ButtonComponent onClick={reverse} id="reverse"
className="e-btn e-info">Reverse Slide Out </ButtonComponent>
        </div>
    </div>
</div>);
}
export default App;
{% endraw %}

```

APP.TSX

```

{% raw %}
import * as React from 'react';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { SidebarComponent, SidebarType } from '@syncfusion/ej2-react-
navigations';
import { enableRipple } from '@syncfusion/ej2-base';
function App() {
    let sidebarObj: SidebarComponent;
    let btnObj: ButtonComponent;
    let showBackdrop: boolean = true;
    function onCreate():void {
        sidebarObj.element.style.visibility='';
    }
}

```

```

    }
    // Zoom Sidebar
    function zoom():void {
        sidebarObj.show();
        sidebarObj.element.classList.add("w3-animate-zoom");
    }
    // Open Door
    function open_door():void {
        sidebarObj.show();
        const sidebar: Element =document.getElementsByClassName("e-sidebar-
overlay")[0];
        sidebar.classList.add("move");
    }
    // Bottom to Top
    function bottom_top(): void {
        sidebarObj.show();
        sidebarObj.element.classList.add("w3-animate-bottom");
    }
    // Rotate
    function rotate(): void {
        sidebarObj.show();
        sidebarObj.element.classList.add("rotate");
    }
    // Rotate 3D Effect
    function rotate_3d(): void {
        sidebarObj.show();
        sidebarObj.element.classList.add("rotate_3d");
    }
    // Reverse Slide Out
    function reverse(): void {
        sidebarObj.show();
        sidebarObj.element.classList.add("reverse_slide_out");
    }
    // Close Sidebar Element
    function close(): void {
        sidebarObj.element.classList.remove("sidebar");
        sidebarObj.element.classList.remove("rotate");
        sidebarObj.element.classList.remove("w3-animate-zoom");
        sidebarObj.element.classList.remove("w3-animate-bottom");
        sidebarObj.element.classList.remove("rotate_3d");
        sidebarObj.element.classList.remove("reverse_slide_out");
        sidebarObj.hide();
    }
    return (
        // Declaration of Sidebar Element
        <div className="control-section">
            <SidebarComponent id="sidebar-element" className="sidebar"
ref={Sidebar => sidebarObj = Sidebar as SidebarComponent}
showBackdrop={showBackdrop} style={{visibility:"hidden"}} created={onCreate}
width="280px">
                <div className="title"> Sidebar content</div>
                <div className="sub-title">
                    * Sidebar is rendered with animation effect
                </div>
                <div className="center-align">
                    { /* Close Button Declaration */ }

```

```

        <ButtonComponent onClick={close} id="close"
className="e-btn close-btn">Close Sidebar</ButtonComponent>
    </div>
</SidebarComponent>
/*End of Sidebar Element Declaration*/
/* Main Content Declaration*/
<div className="content">
    <div className="title">Sidebar Transitions</div>
    <div className="sub-title"> * Click the below button to
render the Sidebar with animation effect.</div>
    <div className="center-align">
        /*Button Declaaration */
        <ButtonComponent onClick={zoom} id="zoom" className="e-
btn e-info">Zoom Sidebar</ButtonComponent>
        <ButtonComponent onClick={open_door} id="open_door"
className="e-btn e-info">Open Door</ButtonComponent>
        <ButtonComponent onClick={bottom_top} id="bottom_top"
className="e-btn e-info">Bottom to Top </ButtonComponent>
    </div>
    <div className="center-align">
        <ButtonComponent onClick={rotate} id="rotate"
className="e-btn e-info">Rotate</ButtonComponent>
        <ButtonComponent onClick={rotate_3d} id="rotate_3d"
className="e-btn e-info">Rotate 3D</ButtonComponent>
        <ButtonComponent onClick={reverse} id="reverse"
className="e-btn e-info">Reverse Slide Out </ButtonComponent>
    </div>
</div>
</div>
)
}
export default App;
{% endraw %}

```

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';
ReactDOM.render(<App />, document.getElementById('sample'));

```