

USER GUIDE

Essential Studio for EJ2 React

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

Syncfusion React UI Components (Essential JS 2).....	9
Components list	9
table	9
How to best read this user guide	11
Getting help	11
See also	11
System Requirements for React Components	12
Browser support	12
React supported versions	12
See also	12
Browser support	12
Required polyfills	13
Using CDN	13
Node.js	13
Getting Started.....	13
Getting Started with Syncfusion React UI Component.....	13
Prerequisites	14
Create the React application.....	14
Add Syncfusion React packages	14
Import the Syncfusion CSS styles	15
Add Syncfusion component to the application.....	15
Run the application	18
Creating a Next.js Application Using Syncfusion React Components	21
What is Next.js?	21
Prerequisites	21
Create a Next.js application	21
Install Syncfusion React packages.....	22
Import Syncfusion CSS styles	22
Add Syncfusion React component	22
Run the application	26
See also	26
Getting Started with React UI Components with Vite	26
Prerequisites	26
Set up the Vite project	26
Add Syncfusion React packages	28

Import Syncfusion CSS styles	28
Add Syncfusion React component	28
Run the project	29
See also	30
Getting Started with Webpack Externals in React.....	30
Prerequisites	30
Create the React application.....	30
Add Syncfusion React packages	30
Add babel packages	31
Add Syncfusion Grid component	31
Install and configure the webpack.....	32
Configuring the webpack externals	34
Configure the package JSON	36
Run the application	36
Getting Started with Create React Remix App with Syncfusion Components.....	36
Prerequisites	36
Benefits to using Remix.....	37
Create Remix application	37
Adding Syncfusion Grid packages	38
Adding CSS reference.....	38
Adding React Grid component.....	38
Run the application	43
Getting Started with React UI Components in the SharePoint Framework	43
Prerequisites	43
Set up the SharePoint project.....	43
Add Syncfusion React packages	44
Import Syncfusion CSS styles	44
Add Syncfusion React component	44
Run the project	46
Getting Started with the Preact Framework with Syncfusion React Components.....	46
Prerequisites	46
Set up the Preact project	46
Add Syncfusion React packages	48
Import Syncfusion CSS styles	48
Add Syncfusion React component	49

Run the project	50
See also	50
Installation	50
Installation	50
Install by using npm CLI.....	50
Install by using package.json.....	51
Npm packages for Syncfusion React UI Components.....	51
Anatomy of npm packages.....	51
Available npm package	52
See also	59
Web Installer	59
Download JavaScript – EJ2 Installer	59
Installation using Web Installer	62
Offline Installer	74
Download JavaScript – EJ2 Installer	74
Installation using Offline Installer	76
Mac Installer	85
Download JavaScript – EJ2 Mac Installer	85
Installing Syncfusion JavaScript – EJ2 Mac Installer.....	89
Linux Installer	94
Download Syncfusion JavaScript Linux Installer	94
Installing Syncfusion JavaScript Linux installer	98
Common Installation Errors	99
Unlocking the license installer using the trial key.....	99
License has expired	99
Unable to find a valid license or trial	100
Unable to install because of another installation.....	102
Unable to install due to controlled folder access	103
Upgrade.....	104
Release History	104
Syncfusion React supported versions	105
React version compatibility.....	105
Syncfusion version information	105
See also	105
Upgrading Syncfusion JavaScript (Essential JS2).....	105

Upgrading to the latest version	105
Upgrade from trial version to license version.....	106
Licensing.....	106
Syncfusion licensing overview	106
Difference between unlock key and license key.....	106
Registering Syncfusion license keys in Build server	107
See also	107
Generate Syncfusion ReactJS License key.....	107
Claim license key	108
See also	109
Register Syncfusion License key in ReactJS application.....	109
Register Syncfusion license key in the project.....	110
Register Syncfusion license key using the npx command.....	110
Activate the license.....	114
See also	115
Syncfusion Licensing Errors.....	115
Licensing errors.....	115
Licensing errors from version 16.2.0* to 20.3.0*	118
Licensing troubleshoot in React.....	122
Is an internet connection required for license validation.....	122
Upgrade from the trial version after purchasing a license	122
Where can I get a License key.....	122
Will the registered license key expire	123
When to generate new license key while upgrading.....	123
License registration for multiple developers on your project	123
Can I use the same key for all the web apps under the project	123
Does the license registration access any resources or data	123
License & Downloads shows the "Essential Studio Enterprise Edition Binary with Test Studio" and the "Project License". Which license to use.....	124
If I registered the license key in both the application and the license text file	124
Reactivating license once after updating the package version while using npx.....	124
Potential causes of licensing errors in applications.	124
Appearance	126
React Themes in Syncfusion Components.....	126
Reference themes in the React application.....	127

Refer themes through npm packages.....	127
Refer themes through CDN reference	129
Common variables	129
See also	145
Size Mode for Syncfusion React Components	145
Size mode for application	145
Size mode for a component.....	146
Change size mode for application at runtime.....	147
Change size mode for a component at runtime	149
Change the font size for all components	151
Change the font family of Syncfusion React components	152
See also	152
React Icons Library	152
Referring icons in the React application	152
Steps to use icons library	153
Available icons	160
See also	160
Overview	160
Customizing theme color from theme studio.....	161
Import previously changed settings into the theme studio	169
Common variables	171
Styled-Component support.....	185
Add styled component to the application	185
Dynamically computed props styling.....	186
Material 3 Theme with CSS Variables	187
Material 3 - Syncfusion React Components.....	187
Common.....	192
Rendering Engine of Syncfusion React Components.....	192
Individual Components Script Dependency.....	193
Accessibility in Syncfusion React Components	195
Accessibility overview	195
Accessibility standards.....	195
Accessibility compliance	196
Ensuring accessibility	196
Accessibility support for specific components.....	197

table	197
Right-To-Left support in Syncfusion React Components	199
Enable RTL for all components	199
Enable RTL for an individual component	200
Animation in React.....	201
Animation effects.....	202
Animation duration.....	203
Animation delay	205
Enable or disable animation globally	207
Drag and Drop in React	207
Draggable	207
Droppable	211
Templates in Syncfusion React Components.....	213
Stateless template	214
Syncfusion React components - Security.....	215
Security Vulnerabilities	215
Security Considerations	215
Globalization	220
Globalization	220
Internationalization.....	221
Getting started with Localization.....	239
State Persistence in Syncfusion React components	242
State Persistence supported components and properties	247
Integration of Material-UI Components	250
Set up the React project.....	250
Integrate Material-UI components	250
Run the project	251
Integration of Syncfusion React Components in Redux Form	252
Create a Redux Form.....	252
Setting up the Redux store	253
Connect with Redux Form.....	253
Add Syncfusion React components.....	253
Form Validation.....	254
Displaying Errors	255
Run the project	255

How To	256
How to resolve Content Security Policy (CSP) errors.....	256
TroubleShoot	256
Content Security Policy	256
See also	257
Visual Studio Code Integration	257
Visual Studio Code Integration	257
Overview	257
Download and Installation.....	258
Prerequisites	258
Install through the Visual Studio Code extensions	258
Install from the Visual Studio Code Marketplace	259
Visual Studio Code Extensions	260
Create project	260
Run the application	263
Add Syncfusion component to the application.....	264
Upgrading the npm packages	265
Visual Studio Integration.....	265
Visual Studio Integration.....	265
Overview	265
Visual Studio Integration.....	265
Create project	265
Convert Project	268
Upgrade Project	271

Syncfusion React UI Components (Essential JS 2)

Syncfusion React (Essential JS 2) library is a modern UI components library that has been built from the ground up to be lightweight, responsive, modular and touch friendly.

[Components list](#)

The Syncfusion React UI components are listed below.

<style>

table

```
{
border:0 !important;
line-height: 2!important;
}
tr
{
border:0 !important;
}
td
{
border:0 !important;
vertical-align: top;
}
.controlanchorlink
{
text-decoration: none!important;
font-size: 14px!important;
text-align: left!important;
padding: 5px 0px;
letter-spacing: 1px;
}
.controlcategory
{
font-size: 14px!important;
text-align: left!important;
font-weight: bold!important;
```

letter-spacing: 0.7px;

}

}

</style>

| | | | |
|------------------------------------|--|--------------------------------------|--------------------------------------|
| | DATA
VISUALIZATION | | DROPDOWNS |
| | | | AutoComplete |
| | | | ListBox |
| GRIDS | Accumulation Chart | CALENDARS | ComboBox |
| DataGrid | Charts | Scheduler | DropDown List |
| Pivot Table | 3D Chart | Gantt Chart | Multiselect DropDown |
| TreeGrid | Stock Chart | Calendar | DropDown Tree |
| Spreadsheet | Circular Gauge | DatePicker | Mention |
| | Linear Gauge | DateRangePicker | |
| FILE VIEWERS &
EDITORS | Maps | DateTime Picker | NAVIGATION |
| | Diagram | TimePicker | |
| Document Editor | HeatMap Chart | | Accordion |
| Image Editor | Map | INPUTS | Carousel |
| In-place Editor | Range Selector | TextBox | Context Menu |
| RichTextEditor | Smith Chart | Input Mask | Menu Bar |
| PDF Viewer | Sparkline Charts | Numeric TextBox | Ribbon |
| Word Processor | Barcode | RadioButton | Sidebar |
| | TreeMap | CheckBox | Tabs |
| LAYOUT | Bullet Chart | Color Picker | Toolbar |
| Dialog | Kanban | File Upload | TreeView |
| ListView | | Range Slider | File Manager |
| Predefined Dialogs | BUTTONS | Toggle Switch Button | Stepper |
| Tooltip | Button | Signature | Breadcrumb |
| Splitter | ButtonGroup | Rating | Pager |
| Dashboard | DropDown Menu | FORMS | AppBar |
| Card | Progress Button | Form Validator | NOTIFICATION |
| Avatar | SplitButton | Query Builder | |
| | Chips | | Toast |
| | Floating Action Button | | Progress Bar |
| | Speed Dial | | Spinner |

| | | | |
|--|--|--|--------------------------|
| | | | Badge |
| | | | Skeleton |
| | | | Message |
| | | | |

How to best read this user guide

- The best way to get started would be to read the "Getting Started" section

of the documentation for the component that you would like to start using first.

The "Getting Started" guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

- Now that you are familiar with the basics of using the component, the next

step would be to start integrating the component into your application.

A good starting point would be to refer to the code snippets in the

[online sample browser](#) which contains

hundreds of code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

- Another valuable resource is the API reference which provides detailed information

on the object hierarchy as well as the settings available on every object.

Getting help

- If you are still not able to find the information that you are looking for at the

self-help resources mentioned above then please contact us by creating a support

ticket in our support site or ask your query in StackOverflow with tag `syncfusion-ej2`.

- Don't see what you need? Please request it in our [feedback portal](#).

Note: Syncfusion does not collect any kind of information when our components are used in customer applications.

See also

- [Product Development Life Cycle](#)
- [Getting Started with Syncfusion React UI Components](#)
- [Project Setup with Visual Studio Code](#)

- [Project Setup with Visual Studio](#)

System Requirements for React Components

This section explains the basic system requirements to work with Syncfusion React UI components.

- React supported version $\geq 15.5.4+$.
- Required [node](#) version $\geq 14.0.0+$, and the supported [npm](#) version will be installed along with the node version. Use the below command to check the node version.

```
`bash
```

```
node --version
```

```
,
```

- For Yarn package manager, the [Yarn version](#) $\geq 0.25+$ is required.

Browser support

The Syncfusion React UI components are supported only in modern browsers. Refer to the [browser compatibility](#) section for more information.

React supported versions

The following table represents the supported React versions by different Syncfusion React UI components releases.

| Version | Syncfusion React components version |
|-----------------------------|-------------------------------------|
| ----- | ----- |
| React v18.0 | 20.2.36 and above |
| React v17.0 | 18.3.50 and above |
| React v16.0 | 16.2.45 and above |

See also

- [React versions](#)
- [React versioning policy](#)

Browser support

The Syncfusion Essential JS 2 components are supported only in modern browsers. This includes the following versions.

| Chrome | Firefox | Opera | Edge | IE | Safari | iOS | Android | Windows Mobile |
|--------|---------|--------|-------|-------|--------|-------|---------|----------------|
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| Latest | Latest | Latest | 13 + | 11 + | 9 + | 9 + | 4.4 + | IE 11 + |

Required polyfills

The following polyfills are required to run Essential JS 2 components in each browser.

| Browser | Polyfills |
|---|-------------|
| Chrome(latest), Firefox(latest), Opera(latest), Edge, Safari 9+ | NONE |
| IE 11 | ES6 Promise |

The Syncfusion Essential JS 2 components are supported in IE 11 browser with ES6 Promise polyfill.

Using CDN

To add ES6 Promise polyfill using a CDN, include this in your HTML file.

```
`ts
<!-- Automatically provides/replaces Promise if missing or broken. -->
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.js"></script>
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.auto.js"></script>
<!-- Minified version of es6-promise-auto below. -->
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.auto.min.js"></script>
`
```

Node.js

ES6 Promise polyfill can also be installed on the node.js.

To install:

```
`ts
yarn add es6-promise
(or)
npm install es6-promise
`
```

To Use:

```
`ts
var Promise = require('es6-promise').Promise;
`
```

For further details, refer to the link [here](#).

Getting Started

Getting Started with Syncfusion React UI Component.

This article provides a step-by-step introduction to get started with Syncfusion React UI components.

Prerequisites

[System requirements for Syncfusion React UI components](#)

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 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
```

```
,
```

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.com](#) public registry. So, choose the component that you want to install.

In this quick start article, the Grid component used as an example. To install the Grid component package, use the following command.

```
`bash
npm install @syncfusion/ej2-react-grids --save
`
or
`bash
yarn add @syncfusion/ej2-react-grids
`
```

You can also checkout the [installation section](#) to know the different ways of installing the packages.

Import the Syncfusion CSS styles

After installing the Syncfusion component packages in your application, import the required themes based on the components used.

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. Let's import the **Material** theme for the Grid component.

Import the CSS styles for the Grid 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-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-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-grids/styles/material.css";
`
```

You can checkout the [Themes topic](#) to know more about built-in themes and different ways to refer themes in React application.

Add Syncfusion component to the application

Start adding the required components to the application. Let's add the Grid component in the **src/App.js** file using the following code.

1. Before adding the Grid component to your markup, import the Grid component in the **src/App.js** file.

```
`bash
import { ColumnDirective, ColumnsDirective, GridComponent } from '@syncfusion/ej2-react-grids';
`
```

2. Then, to display the Grid with records, add the Grid component and bind the `dataSource` to it. Here, we have mapped the simple data as the `dataSource`.

```
`js
import { ColumnDirective, ColumnsDirective, GridComponent } from '@syncfusion/ej2-react-grids';
import './App.css';
function App() {
  const data = [
    {
      OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new Date(8364186e5),
      ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59 rue de l Abbaye',
      ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France', Freight: 32.38, Verified: !0
    },
    {
      OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new Date(836505e6),
      ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress: 'Luisenstr. 48',
      ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight: 11.61, Verified: !1
    },
    {
      OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new Date(8367642e5),
      ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
      ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 65.83, Verified: !0
    }
  ];
  return (
    <GridComponent dataSource={data}>
      <ColumnsDirective>
        <ColumnDirective field='OrderID' width='100' textAlign="Right"/>
        <ColumnDirective field='CustomerID' width='100'/>
        <ColumnDirective field='EmployeeID' width='100' textAlign="Right"/>
        <ColumnDirective field='Freight' width='100' format="C2" textAlign="Right"/>
        <ColumnDirective field='ShipCountry' width='100'/>
      </ColumnsDirective>
    </GridComponent>
  );
}
```



```
);
}
export default App;
`ts
import { ColumnDirective, ColumnsDirective, GridComponent } from '@syncfusion/ej2-react-grids';
import './App.css';
function App() {
  const data = [
    {
      OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new Date(8364186e5),
      ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59 rue de l Abbaye',
      ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France', Freight: 32.38, Verified: !0
    },
    {
      OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new Date(836505e6),
      ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress: 'Luisenstr. 48',
      ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight: 11.61, Verified: !1
    },
    {
      OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new Date(8367642e5),
      ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
      ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 65.83, Verified: !0
    }
  ];
  return (<GridComponent dataSource={data}>
    <ColumnsDirective>
      <ColumnDirective field='OrderID' width='100' textAlign="Right"/>
      <ColumnDirective field='CustomerID' width='100'/>
      <ColumnDirective field='EmployeeID' width='100' textAlign="Right"/>
      <ColumnDirective field='Freight' width='100' format="C2" textAlign="Right"/>
      <ColumnDirective field='ShipCountry' width='100'/>
    </ColumnsDirective>
```

```

</GridComponent>;
}
export default App;
`

```

Run the application

All are set. Now, run the application using the following command.

```

`bash
npm start
`
or
`bash
yarn start
`

```

The output will appears as follows.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ColumnDirective, ColumnsDirective, GridComponent } from
'@syncfusion/ej2-react-grids';
function App() {
  const data = [
    {
      OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate:
new Date(8364186e5),
      ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims',
      ShipAddress: '59 rue de l Abbaye',
      ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry:
'France', Freight: 32.38, Verified: !0
    },
    {
      OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate:
new Date(836505e6),
      ShipName: 'Toms Spezialitäten', ShipCity: 'Münster',
      ShipAddress: 'Luisenstr. 48',
      ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry:
'Germany', Freight: 11.61, Verified: !1
    },
    {
      OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate:
new Date(8367642e5),
      ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro',
      ShipAddress: 'Rua do Paço, 67',
      ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry:
'Brazil', Freight: 65.83, Verified: !0
    }
  ];
}

```

```

    return (<GridComponent dataSource={data}>
      <ColumnsDirective>
        <ColumnDirective field='OrderID' width='100'
textAlign="Right"/>
        <ColumnDirective field='CustomerID' width='100' />
        <ColumnDirective field='EmployeeID' width='100'
textAlign="Right"/>
        <ColumnDirective field='Freight' width='100' format="C2"
textAlign="Right"/>
        <ColumnDirective field='ShipCountry' width='100' />
      </ColumnsDirective>
    </GridComponent>);
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ColumnDirective, ColumnsDirective, GridComponent } from
'@syncfusion/ej2-react-grids';
function App() {
  const data = [
    {
      OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new
Date(8364186e5),
      ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims',
      ShipAddress: '59 rue de l Abbaye',
      ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France',
      Freight: 32.38, Verified: !0
    },
    {
      OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new
Date(836505e6),
      ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress:
'Luisenstr. 48',
      ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany',
      Freight: 11.61, Verified: !1
    },
    {
      OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new
Date(8367642e5),
      ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro',
      ShipAddress: 'Rua do Paço, 67',
      ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry:
'Brazil', Freight: 65.83, Verified: !0
    }
  ];
  return (
    <GridComponent dataSource={data}>
      <ColumnsDirective>
        <ColumnDirective field='OrderID' width='100'
textAlign="Right"/>
        <ColumnDirective field='CustomerID' width='100' />

```

```

        <ColumnDirective field='EmployeeID' width='100'
textAlign="Right"/>
        <ColumnDirective field='Freight' width='100' format="C2"
textAlign="Right"/>
        <ColumnDirective field='ShipCountry' width='100' />
    </ColumnsDirective>
</GridComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Button</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-
grids/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://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"></sc
ript>
    <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='sample'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

To know more functionality about the Grid component, refer to the [Grid component](#) section.

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 components.

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 18.17](#) 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-grid**.

CMD

```
√ What is your project named? » ej2-nextjs-grid
```

2. Select the required packages.

CMD

```
√ What is your project named? ... ej2-nextjs-grid
√ 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-grid.
```

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

CMD

```
cd ej2-nextjs-grid
```

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 Grid component](#) is used as an example. To install the React Grid component in the project, use the following command:

NPM

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

YARN

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

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-
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-
grids/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 Grid component to the Next.js project:

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

DATASOURCE.TSX

```
export let data: Object[] = [
{
  OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new
  Date(8364186e5),
  ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59
  rue de l Abbaye',
  ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France', Freight:
  32.38, Verified: !0
},
{
  OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new
  Date(836505e6),
  ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress:
  'Luisenstr. 48',
  ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight:
  11.61, Verified: !1
},
{
  OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new
  Date(8367642e5),
  ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do
  Paço, 67',
  ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil',
  Freight: 65.83, Verified: !0
},
{
  OrderID: 10251, CustomerID: 'VICTE', EmployeeID: 3, OrderDate: new
  Date(8367642e5),
  ShipName: 'Victuailles en stock', ShipCity: 'Lyon', ShipAddress: '2, rue du
  Commerce',
  ShipRegion: 'CJ', ShipPostalCode: '69004', ShipCountry: 'France', Freight:
  41.34, Verified: !0
},
{
  OrderID: 10252, CustomerID: 'SUPRD', EmployeeID: 4, OrderDate: new
  Date(8368506e5),
  ShipName: 'Suprêmes délices', ShipCity: 'Charleroi', ShipAddress: 'Boulevard
  Tirou, 255',
  ShipRegion: 'CJ', ShipPostalCode: 'B-6000', ShipCountry: 'Belgium', Freight:
  51.3, Verified: !0
},
{
  OrderID: 10253, CustomerID: 'HANAR', EmployeeID: 3, OrderDate: new
  Date(836937e6),
  ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do
  Paço, 67',
  ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil',
  Freight: 58.17, Verified: !0
},
{
  OrderID: 10254, CustomerID: 'CHOPS', EmployeeID: 5, OrderDate: new
  Date(8370234e5),
  ShipName: 'Chop-suey Chinese', ShipCity: 'Bern', ShipAddress: 'Hauptstr.
  31',
```

```

ShipRegion: 'CJ', ShipPostalCode: '3012', ShipCountry: 'Switzerland',
Freight: 22.98, Verified: !1
},
{
OrderID: 10255, CustomerID: 'RICSU', EmployeeID: 9, OrderDate: new
Date(8371098e5),
ShipName: 'Richter Supermarkt', ShipCity: 'Genève', ShipAddress: 'Starenweg
5',
ShipRegion: 'CJ', ShipPostalCode: '1204', ShipCountry: 'Switzerland',
Freight: 148.33, Verified: !0
},
{
OrderID: 10256, CustomerID: 'WELLI', EmployeeID: 3, OrderDate: new
Date(837369e6),
ShipName: 'Wellington Importadora', ShipCity: 'Resende', ShipAddress: 'Rua
do Mercado, 12',
ShipRegion: 'SP', ShipPostalCode: '08737-363', ShipCountry: 'Brazil',
Freight: 13.97, Verified: !1
},
{
OrderID: 10257, CustomerID: 'HILAA', EmployeeID: 4, OrderDate: new
Date(8374554e5),
ShipName: 'HILARION-Abastos', ShipCity: 'San Cristóbal', ShipAddress:
'Carrera 22 con Ave. Carlos Soublette #8-35',
ShipRegion: 'Táchira', ShipPostalCode: '5022', ShipCountry: 'Venezuela',
Freight: 81.91, Verified: !0
},
{
OrderID: 10258, CustomerID: 'ERNSH', EmployeeID: 1, OrderDate: new
Date(8375418e5),
ShipName: 'Ernst Handel', ShipCity: 'Graz', ShipAddress: 'Kirchgasse 6',
ShipRegion: 'CJ', ShipPostalCode: '8010', ShipCountry: 'Austria', Freight:
140.51, Verified: !0
},
{
OrderID: 10259, CustomerID: 'CENTC', EmployeeID: 4, OrderDate: new
Date(8376282e5),
ShipName: 'Centro comercial Moctezuma', ShipCity: 'México D.F.',
ShipAddress: 'Sierras de Granada 9993',
ShipRegion: 'CJ', ShipPostalCode: '05022', ShipCountry: 'Mexico', Freight:
3.25, Verified: !1
},
{
OrderID: 10260, CustomerID: 'OTTIK', EmployeeID: 4, OrderDate: new
Date(8377146e5),
ShipName: 'Ottilies Käseladen', ShipCity: 'Köln', ShipAddress:
'Mehrheimerstr. 369',
ShipRegion: 'CJ', ShipPostalCode: '50739', ShipCountry: 'Germany', Freight:
55.09, Verified: !0
},
{
OrderID: 10261, CustomerID: 'QUEDE', EmployeeID: 4, OrderDate: new
Date(8377146e5),
ShipName: 'Que Delícia', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua da
Panificadora, 12',
ShipRegion: 'RJ', ShipPostalCode: '02389-673', ShipCountry: 'Brazil',
Freight: 3.05, Verified: !1

```



```

},
{
  OrderID: 10262, CustomerID: 'RATTC', EmployeeID: 8, OrderDate: new
  Date(8379738e5),
  ShipName: 'Rattlesnake Canyon Grocery', ShipCity: 'Albuquerque',
  ShipAddress: '2817 Milton Dr.',
  ShipRegion: 'NM', ShipPostalCode: '87110', ShipCountry: 'USA', Freight:
  48.29, Verified: !0
}
];

```

2. Then, import and define the Grid component with the [dataSource](#) property and column definitions in the **src/app/page.tsx** file, as shown below:

PAGE.TSX

```

'use client'
import {
  ColumnDirective, ColumnsDirective, GridComponent,
  Inject, Page, Sort, Filter, Group
} from '@syncfusion/ej2-react-grids';
import { data } from '../datasource';
export default function Home() {
  const pageSettings: object = { pageSize: 6 };
  const filterSettings: object = { type: 'Excel' };
  return (
    <>
    <h2>Syncfusion React Grid Component</h2>
    <GridComponent
      dataSource={data}
      allowGrouping={true}
      allowSorting={true}
      allowFiltering={true}
      allowPaging={true}
      pageSettings={pageSettings}
      filterSettings={filterSettings}
      height={180}
    >
    <ColumnsDirective>
    <ColumnDirective field="OrderID" width="100" textAlign="Right" />
    <ColumnDirective field="CustomerID" width="100" />
    <ColumnDirective field="EmployeeID" width="100" textAlign="Right" />
    <ColumnDirective
      field="Freight"
      width="100"
      format="C2"
      textAlign="Right"
    />
    <ColumnDirective field="ShipCountry" width="100" />
    </ColumnsDirective>
    <Inject services={[Page, Sort, Filter, Group]} />
    </GridComponent>
    </>
  )
}

```

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 Grid component, refer to the [documentation](#).

[View the NEXT.js Grid sample in the GitHub repository.](#)

See also

[Getting Started with the Syncfusion React UI Component](#)

Getting Started with React UI Components with Vite

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion React components.

Vite is a build tool and development server for modern web projects. Vite is designed to be fast and lightweight, supporting modern web technologies such as ES modules, TypeScript, JSX, and CSS modules. Vite's development server uses native ES modules in modern browsers, providing faster startup times and quicker feedback loops during development.

Prerequisites

[System requirements for Syncfusion React UI components](#)

Set up the Vite project

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

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

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
```

```
? Project name: » my-project
```

```
,
```

2. Select **React** as the framework. It will create a React project.

```
`bash
```

? Select a framework: » - Use arrow-keys. Return to submit.

Vanilla

Vue

React

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as framework variant to build this Vite project using JavaScript and React.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

JavaScript + SWC

TypeScript + SWC

,

4. Upon completing the aforementioned steps to create **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

Add 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 Grid component](#) as an example. To use the React Grid component in the project, the `@syncfusion/ej2-react-grids` package needs to be installed using the following command:

```
`bash
```

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

```
`
```

or

```
`bash
```

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

```
`
```

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, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Grid component and its dependents were imported into the `src/App.css` file.

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-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-grids/styles/material.css";
```

The order of importing CSS styles should be in line with its dependency graph.

Add Syncfusion React component

Follow the below steps to add the React Grid component to the Vite project:

1.Before adding the Grid component to your markup, import the Grid component in the `src/App.jsx` file.

APP.JSX

```
import { ColumnDirective, ColumnsDirective, GridComponent } from
 '@syncfusion/ej2-react-grids';
```

2.Then, define the Grid component with the [dataSource](#) property and column definitions. Declare the values for the `dataSource` property.

APP.JSX

```
import './App.css'
import { ColumnDirective, ColumnsDirective, GridComponent } from
 '@syncfusion/ej2-react-grids';
function App() {
  const data = [
    {
      OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
      Freight: 32.38
    },
    {
      OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
      Freight: 11.61
    },
    {
      OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
      Freight: 65.83
    }
  ];
  return (
    <GridComponent dataSource={data}>
      <ColumnsDirective>
        <ColumnDirective field='OrderID' width='100' textAlign="Right"/>
        <ColumnDirective field='CustomerID' width='100' />
        <ColumnDirective field='EmployeeID' width='100' textAlign="Right"/>
        <ColumnDirective field='Freight' width='100' format="C2" textAlign="Right"/>
        <ColumnDirective field='ShipCountry' width='100' />
      </ColumnsDirective>
    </GridComponent>
  );
}
export default 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:

| OrderID | CustomerID | EmployeeID | Freight | ShipCountry |
|---------|------------|------------|---------|-------------|
| 10248 | VINET | 5 | \$32.38 | France |
| 10249 | TOMSP | 6 | \$11.61 | Germany |
| 10250 | HANAR | 4 | \$65.83 | Brazil |

See also

[Getting Started with the Syncfusion React UI Component](#)

[Getting Started with Webpack Externals in React](#)

The [Webpack Externals](#) are used to ignore the packages from the application while bundling and adding them as external CDN script references.

This article provides a step-by-step introduction to creating a simple react application and configuring the Syncfusion react packages with [Externals](#).

Prerequisites

[System requirements for Syncfusion React UI components](#)

Create the React application

- Open the command prompt in your desired location. To create the application folder, use the following commands.

```
`bash
```

```
mkdir Syncfusion-react-demo
```

```
cd Syncfusion-react-demo
```

```
`
```

- Use the following commands to create a `package.json` file. Then install the [react](#) and [react-dom](#) packages in the application.

```
`bash
```

```
npm init
```

```
npm install react react-dom
```

```
`
```

[Add Syncfusion React packages](#)

Install the required Syncfusion React component package in the application. All Syncfusion React (Essential JS 2) packages are published on the [npmjs.com](#) public registry. So, choose the component that you want to install.

In this article, the Grid component is used as an example. To install the Grid component package, use the following command.

```
`bash
```

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

Add babel packages

Install the required babel presets packages in the application.

```
`bash
```

```
npm install babel-loader @babel/preset-react @babel/preset-env @babel/core --save-dev
```

Add Syncfusion Grid component

- Create an `src` folder in the application repository. Then create an `index.html` Html, `index.js` entry and `App.js` component files inside the `src` folder location. Start adding the required components to the application.
- Open the `src/App.js` file and add the Grid component as follows.

```
`javascript
```

```
import React from "react";
```

```
import { ColumnDirective, ColumnsDirective, GridComponent } from '@syncfusion/ej2-react-grids';
```

```
function App() {
```

```
  const data = [
```

```
    {
```

```
      OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new Date(8364186e5),
```

```
      ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59 rue de l Abbaye',
```

```
      ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France', Freight: 32.38, Verified: !0
```

```
    },
```

```
    {
```

```
      OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new Date(836505e6),
```

```
      ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress: 'Luisenstr. 48',
```

```
      ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight: 11.61, Verified: !1
```

```
    },
```

```
    {
```

```
      OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new Date(8367642e5),
```

```
      ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
```

```
      ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 65.83, Verified: !0
```

```
    }
```

```
  ];
```

```
  return (
```

```

<GridComponent dataSource={data}>
  <ColumnsDirective>
    <ColumnDirective field='OrderID' width='100' textAlign="Right"/>
    <ColumnDirective field='CustomerID' width='100'/>
    <ColumnDirective field='EmployeeID' width='100' textAlign="Right"/>
    <ColumnDirective field='Freight' width='100' format="C2" textAlign="Right"/>
    <ColumnDirective field='ShipCountry' width='100'/>
  </ColumnsDirective>
</GridComponent>
);
}
export default App;
`

```

- Open the `src/index.js` file and add the app component as follows.

```

`js
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <App />
);
`

```

Install and configure the webpack

- Install the webpack packages by using the following command.

```

`bash
npm install webpack webpack-cli webpack-dev-server html-webpack-plugin --save-dev
`

```

- Create a `webpack.config.js` file in the application root repository using the following code sample.

```

`js

```



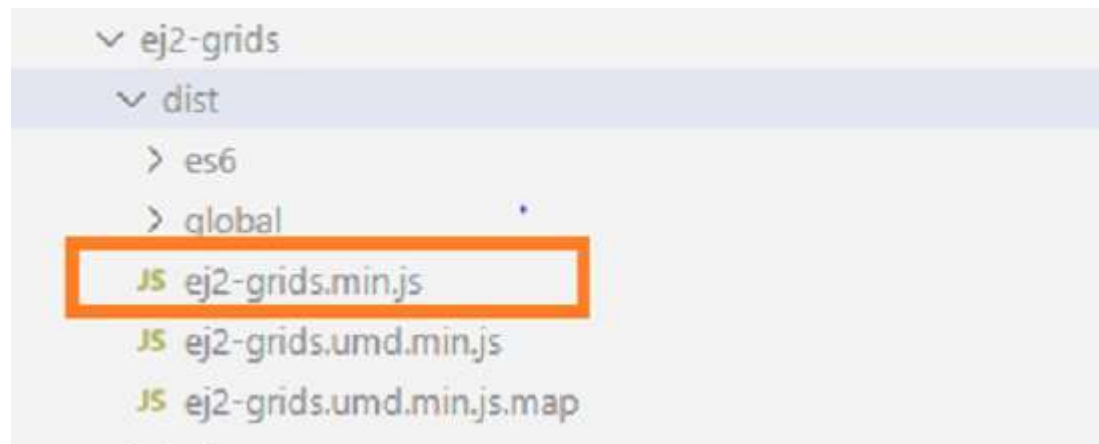
```
const path = require('path');
const HTMLWebpackPlugin = require('html-webpack-plugin');
module.exports = {
  entry: path.resolve(dirname, "src/index.js"),
  output: {
    path: path.join(dirname, '/build'),
    filename: 'bundles.js',
    libraryTarget: 'umd',
  },
  plugins: [
    new HTMLWebpackPlugin({
      template: './src/index.html'
    }),
  ],
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: "babel-loader",
          options: {
            presets: ["@babel/preset-env", "@babel/preset-react"]
          }
        }
      },
    ]
  }
},
```

Configuring the webpack externals

- The Syncfusion react library name in the externals should be `SyncfusionReact[PackageName]`. The `PackageName` should start with a capital letter for every hyphen like `SyncfusionReactBarcodeGenerator` for `ej2-react-barcode-generator` package.
- Open the `webpack.config.js` file and add the Syncfusion react packages in the externals option as follows.

```
`js
module.exports = {
  externals: {
    "react": "React",
    "react-dom": "ReactDOM",
    "@syncfusion/ej2-react-grids": "SyncfusionReactGrids"
  },
}
```

- Find the Syncfusion external CDN script `[package-name].min.js` file inside `dist` folder of the packages.



- Add the [React](#) and Syncfusion react Grid CDN references in the `src/index.html` file. The order of individual Syncfusion control package loading should be in line with its dependency graph. The CDN of the Dependency Packages should be included manually before the intended individual Syncfusion control package CDN.

```
`html
<!DOCTYPE html>
<html lang="en">
```

```
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<!-- React and ReactDOM scripts -->
<script crossorigin src="https://unpkg.com/react@18.2.0/umd/react.production.min.js"></script>
<script crossorigin src="https://unpkg.com/react-dom@18.2.0/umd/react-dom.production.min.js"></script>
<!-- Syncfusion React controls styles -->
<link rel="stylesheet" href="https://cdn.syncfusion.com/ej2/material.css" />
<!-- Syncfusion React controls scripts -->
<script src="https://cdn.syncfusion.com/ej2/ej2-base/dist/ej2-base.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-data/dist/ej2-data.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-buttons/dist/ej2-buttons.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-splitbuttons/dist/ej2-splitbuttons.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-popups/dist/ej2-popups.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-navigations/dist/ej2-navigations.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-inputs/dist/ej2-inputs.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-dropdowns/dist/ej2-dropdowns.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-calendars/dist/ej2-calendars.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-lists/dist/ej2-lists.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-excel-export/dist/ej2-excel-export.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-pdf-export/dist/ej2-pdf-export.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-file-utils/dist/ej2-file-utils.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-compression/dist/ej2-compression.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-grids/dist/ej2-grids.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-react-base/dist/ej2-react-base.min.js"></script>
<script src="https://cdn.syncfusion.com/ej2/ej2-react-grids/dist/ej2-react-grids.min.js"></script>
<title>Document</title>
</head>
<body>
<div id="root"></div>
</body>
```

```
</html>
```

Refer to the [Script Dependency](#) section to discover the correct script reference sequence and its dependencies for a certain Syncfusion React component.

Configure the package JSON

Open the `package.json` file and configure the application process in the scripts as follows.

```
{
  "scripts": {
    "start": "webpack-dev-server --mode development --open --hot",
    "build": "webpack --mode production"
  }
}
```

Run the application

Run the application using the following command.

```
`bash
npm start
```

OrderID	CustomerID	EmployeeID	Freight	ShipCountry
10248	VINET	5	\$32.38	France
10249	TOMSP	6	\$11.61	Germany
10250	HANAR	4	\$65.83	Brazil

[View Sample in GitHub.](#)

Getting Started with Create React Remix App with Syncfusion Components

This document helps you to create a simple Remix application with React Framework and Syncfusion React components.

Prerequisites

Before getting started with Syncfusion React Components with Remix project, check whether the following are installed in the developer machine.

- Node.js 14 or greater
- npm 7 or greater

Benefits to using Remix

Nested pages

The pages inside `./route` folder is nested in the route where you can embed these components into the parent page. It reduces the loading time of the page.

Error boundaries

If you get an error in a nested route or a Remix component, the errors are limited and the component will fail to render, but it doesn't break the entire page.

Transition

Remix automatically handles all loading states, you have to do is tell Remix what to show when the app is loading.

Create Remix application

Install [create-remix](#) by running the following command.

,

```
npm install -g create-remix
```

,

To setup basic Remix sample, use the following commands.

,

```
npx create-remix@latest
```

,

`create-remix@latest` command create the remix app with latest version of packages.

When you run this command, You will be asked the following questions.

,

Where would you like to create your app? quick-start

What type of app do you want to create? Just the basics

Where do you want to deploy? Choose Remix if you're unsure; it's easy to change deployment targets.
Remix App Server

TypeScript or JavaScript? Typescript

Do you want me to run `npm install`? Yes

,

- The Remix app is created in **quick-start** folder.
- There are two types of app creation in remix such as A pre-configured stack ready for production and Just the Basics. For this sample **Just the Basics** is chosen.
- There are a number of servers to choose. We use the default, Remix App Server.
- There is also a choice between TypeScript and JavaScript. In this example Typescript is used.

After app created, use the below command to step inside app folder.

,

```
cd quick-start
```

```
,
```

Adding Syncfusion Grid packages

All the available Syncfusion React packages are published in [npmjs.com](https://www.npmjs.com) public registry. To install React Grid component, use the following command

```
,
```

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

```
,
```

The --save will instruct NPM to include the Grid package inside of the dependencies section of the `package.json`.

Adding CSS reference

Reference themes for Syncfusion components in the `app/route.tsx` as CSS reference.

```
`ts
```

```
export let links = () => {
```

```
  return [{ rel: "stylesheet", href: "https://cdn.syncfusion.com/ej2/material.css" }];
```

```
};
```

```
,
```

Adding React Grid component

Now, you can add Syncfusion React components in the Remix application. For getting started, add the React Grid component in `app/route/index.tsx` file using following code.

```
`ts
```

```
import { ColumnDirective, ColumnsDirective, GridComponent, Inject, Page, Sort } from  
'@syncfusion/ej2-react-grids';
```

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

```
import { data } from '../datasource';
```

```
export let meta = () => {
```

```
  return {
```

```
    title: "Syncfusion Grid Remix",
```

```
    description: "Syncfusion Grid components with Remix",
```

```
  };
```

```
};
```

```
export default function Index() {
```

```
  return (
```

```
    <<<GridComponent dataSource={data} allowPaging={true}>
```

```
      <ColumnsDirective>
```

```

<ColumnDirective field="OrderID" width="100" textAlign="Right" />
<ColumnDirective field="CustomerID" width="100" />
<ColumnDirective field="EmployeeID" width="100" textAlign="Right" />
<ColumnDirective field="Freight" width="100" format="C2" textAlign="Right"/>
<ColumnDirective field="ShipCountry" width="100" />
</ColumnsDirective>
<Inject services={[Page, Sort]} />
</GridComponent></>
);
}
`

```

Create a data source file in `app/datasource.tsx` and add the following data for the grid component.

```

`ts
export let data: Object[] = [
{
  OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new Date(8364186e5),
  ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59 rue de l Abbaye',
  ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France', Freight: 32.38, Verified: !0
},
{
  OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new Date(836505e6),
  ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress: 'Luisenstr. 48',
  ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight: 11.61, Verified: !1
},
{
  OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new Date(8367642e5),
  ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
  ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 65.83, Verified: !0
},
{
  OrderID: 10251, CustomerID: 'VICTE', EmployeeID: 3, OrderDate: new Date(8367642e5),
  ShipName: 'Victuailles en stock', ShipCity: 'Lyon', ShipAddress: '2, rue du Commerce',
  ShipRegion: 'CJ', ShipPostalCode: '69004', ShipCountry: 'France', Freight: 41.34, Verified: !0
}
]

```

```
},
{
  OrderID: 10252, CustomerID: 'SUPRD', EmployeeID: 4, OrderDate: new Date(8368506e5),
  ShipName: 'Suprêmes délices', ShipCity: 'Charleroi', ShipAddress: 'Boulevard Tirou, 255',
  ShipRegion: 'CJ', ShipPostalCode: 'B-6000', ShipCountry: 'Belgium', Freight: 51.3, Verified: !0
},
{
  OrderID: 10253, CustomerID: 'HANAR', EmployeeID: 3, OrderDate: new Date(836937e6),
  ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
  ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 58.17, Verified: !0
},
{
  OrderID: 10254, CustomerID: 'CHOPS', EmployeeID: 5, OrderDate: new Date(8370234e5),
  ShipName: 'Chop-suey Chinese', ShipCity: 'Bern', ShipAddress: 'Hauptstr. 31',
  ShipRegion: 'CJ', ShipPostalCode: '3012', ShipCountry: 'Switzerland', Freight: 22.98, Verified: !1
},
{
  OrderID: 10255, CustomerID: 'RICSU', EmployeeID: 9, OrderDate: new Date(8371098e5),
  ShipName: 'Richter Supermarkt', ShipCity: 'Genève', ShipAddress: 'Starenweg 5',
  ShipRegion: 'CJ', ShipPostalCode: '1204', ShipCountry: 'Switzerland', Freight: 148.33, Verified: !0
},
{
  OrderID: 10256, CustomerID: 'WELLI', EmployeeID: 3, OrderDate: new Date(837369e6),
  ShipName: 'Wellington Importadora', ShipCity: 'Resende', ShipAddress: 'Rua do Mercado, 12',
  ShipRegion: 'SP', ShipPostalCode: '08737-363', ShipCountry: 'Brazil', Freight: 13.97, Verified: !1
},
{
  OrderID: 10257, CustomerID: 'HILAA', EmployeeID: 4, OrderDate: new Date(8374554e5),
  ShipName: 'HILARION-Abastos', ShipCity: 'San Cristóbal', ShipAddress: 'Carrera 22 con Ave. Carlos
  Soublette #8-35',
  ShipRegion: 'Táchira', ShipPostalCode: '5022', ShipCountry: 'Venezuela', Freight: 81.91, Verified: !0
},
{
```



```
OrderID: 10258, CustomerID: 'ERNSH', EmployeeID: 1, OrderDate: new Date(8375418e5),
ShipName: 'Ernst Handel', ShipCity: 'Graz', ShipAddress: 'Kirchgasse 6',
ShipRegion: 'CJ', ShipPostalCode: '8010', ShipCountry: 'Austria', Freight: 140.51, Verified: !0
},
{
OrderID: 10259, CustomerID: 'CENTC', EmployeeID: 4, OrderDate: new Date(8376282e5),
ShipName: 'Centro comercial Moctezuma', ShipCity: 'México D.F.', ShipAddress: 'Sierras de Granada
9993',
ShipRegion: 'CJ', ShipPostalCode: '05022', ShipCountry: 'Mexico', Freight: 3.25, Verified: !1
},
{
OrderID: 10260, CustomerID: 'OTTIK', EmployeeID: 4, OrderDate: new Date(8377146e5),
ShipName: 'Ottilies Käseladen', ShipCity: 'Köln', ShipAddress: 'Mehrheimerstr. 369',
ShipRegion: 'CJ', ShipPostalCode: '50739', ShipCountry: 'Germany', Freight: 55.09, Verified: !0
},
{
OrderID: 10261, CustomerID: 'QUEDE', EmployeeID: 4, OrderDate: new Date(8377146e5),
ShipName: 'Que Delícia', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua da Panificadora, 12',
ShipRegion: 'RJ', ShipPostalCode: '02389-673', ShipCountry: 'Brazil', Freight: 3.05, Verified: !1
},
{
OrderID: 10262, CustomerID: 'RATTC', EmployeeID: 8, OrderDate: new Date(8379738e5),
ShipName: 'Rattlesnake Canyon Grocery', ShipCity: 'Albuquerque', ShipAddress: '2817 Milton Dr.',
ShipRegion: 'NM', ShipPostalCode: '87110', ShipCountry: 'USA', Freight: 48.29, Verified: !0
}};
export let sdata: Object[] = [
{
OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new Date(8364186e5),
ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59 rue de l Abbaye',
ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'Brazil', Freight: 32.38, Verified: !0
},
{
OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new Date(836505e6),
```

```
ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress: 'Luisenstr. 48',
ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight: 11.61, Verified: !1
},
{
  OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new Date(8367642e5),
  ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
  ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 65.83, Verified: !0
},
{
  OrderID: 10251, CustomerID: 'VICTE', EmployeeID: 3, OrderDate: new Date(8367642e5),
  ShipName: 'Victuailles en stock', ShipCity: 'Lyon', ShipAddress: '2, rue du Commerce',
  ShipRegion: 'CJ', ShipPostalCode: '69004', ShipCountry: 'France', Freight: 41.34, Verified: !0
},
{
  OrderID: 10252, CustomerID: 'SUPRD', EmployeeID: 4, OrderDate: new Date(8368506e5),
  ShipName: 'Suprêmes délices', ShipCity: 'Charleroi', ShipAddress: 'Boulevard Tirou, 255',
  ShipRegion: 'CJ', ShipPostalCode: 'B-6000', ShipCountry: 'Belgium', Freight: 51.3, Verified: !0
},
{
  OrderID: 10253, CustomerID: 'HANAR', EmployeeID: 3, OrderDate: new Date(836937e6),
  ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
  ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 58.17, Verified: !0
},
{
  OrderID: 10254, CustomerID: 'CHOPS', EmployeeID: 5, OrderDate: new Date(8370234e5),
  ShipName: 'Chop-suey Chinese', ShipCity: 'Bern', ShipAddress: 'Hauptstr. 31',
  ShipRegion: 'CJ', ShipPostalCode: '3012', ShipCountry: 'Brazil', Freight: 22.98, Verified: !1
},
{
  OrderID: 10255, CustomerID: 'RICSU', EmployeeID: 9, OrderDate: new Date(8371098e5),
  ShipName: 'Richter Supermarkt', ShipCity: 'Genève', ShipAddress: 'Starenweg 5',
  ShipRegion: 'CJ', ShipPostalCode: '1204', ShipCountry: 'Switzerland', Freight: 148.33, Verified: !0
}
});
```

Run the application

Run your Remix application in development mode using `run dev` command,

```
npm run dev
```

For deployment, build your app for production,

```
npm run build
```

Then run the app in production mode:

```
npm run start
```

[View source code](#)

Getting Started with React UI Components in the SharePoint Framework

This article provides a step-by-step guide for setting up a [SharePoint](#) project and integrating the Syncfusion React components.

SharePoint Framework (SPFx) is a development model and framework provided by Microsoft for building custom solutions and extensions for SharePoint and Microsoft Teams. It is a modern, client-side framework that allows developers to create web parts, extensions, and customizations that can be deployed and used within SharePoint sites and Teams applications.

Prerequisites

- [System requirements for Syncfusion React UI components](#)
- [System requirements for the SharePoint Framework Development Environment](#)

Set up the SharePoint project

Create a new SPFx project using the following command:

1\ To initiate the creation of a new [SharePoint](#) project, use the following commands:

```
`bash
```

```
yo @microsoft/sharepoint
```

2\ Let's specify the name of the project as `my-project` and the name of the WebPart as `App` for this article. To set up additional configurations based on your requirements, it will direct you to a series of questions for the project as below:

```
`bash
```

Let's create a new Microsoft 365 solution.

? What is your solution name? my-project

? Which type of client-side component to create? WebPart

Add new Web part to solution my-project.

? What is your Web part name? App

? Which template would you like to use? React

,

3\.. To establish trust for the certificate in the development environment, execute the provided command:

```
`bash
```

```
gulp trust-dev-cert
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

Add 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 Grid component](#) as an example. To use the React Grid component in the project, the **@syncfusion/ej2-react-grids** package needs to be installed using the following command:

```
`bash
```

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

,

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 the [themes topic](#) to learn more about built-in themes and different ways to refer to themes in a React project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary Material CSS styles for the Grid component were imported into the **~/src/webparts/app/components/App.tsx** file.

APP.TSX

```
require ('@syncfusion/ej2-react-grids/styles/material.css');
```

Add Syncfusion React component

Follow the below steps to add the React Grid component:

1\.. In **App.tsx** file inside the **~/src/webparts/app/components** folder, declare the values for the [dataSource](#) property.

APP.TSX

```
const data = [
  {
    OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
    Freight: 32.38
  },
  {
    OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
    Freight: 11.61
  },
  {
    OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
    Freight: 65.83
  }
];
```

2\ Define the Grid component with the `dataSource` property and column definitions.

APP.TSX

```
import { ColumnDirective, ColumnsDirective, GridComponent } from
 '@syncfusion/ej2-react-grids';
export default class App extends React.Component<IAppProps, {}> {
  public render(): React.ReactElement<IAppProps> {
    return (
      <GridComponent dataSource={data}>
        <ColumnsDirective>
          <ColumnDirective field='OrderID' width='100' textAlign="Right" />
          <ColumnDirective field='CustomerID' width='100' />
          <ColumnDirective field='EmployeeID' width='100' textAlign="Right" />
          <ColumnDirective field='Freight' width='100' format="C2" textAlign="Right"
            />
          <ColumnDirective field='ShipCountry' width='100' />
        </ColumnsDirective>
      </GridComponent>
    );
  }
}
```

Here is the summarized code for the above steps:

APP.TSX

```
import * as React from 'react';
import { IAppProps } from './IAppProps';
import { ColumnDirective, ColumnsDirective, GridComponent } from
 '@syncfusion/ej2-react-grids';
require('@syncfusion/ej2-react-grids/styles/material.css')
export default class App extends React.Component<IAppProps, {}> {
  public render(): React.ReactElement<IAppProps> {
    const data = [
      {
        OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
        Freight: 32.38
      },
      {

```

```
OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',  
Freight: 11.61  
},  
{  
OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',  
Freight: 65.83  
}  
];  
return (  
<GridComponent dataSource={data}>  
<ColumnsDirective>  
<ColumnDirective field='OrderID' width='100' textAlign="Right" />  
<ColumnDirective field='CustomerID' width='100' />  
<ColumnDirective field='EmployeeID' width='100' textAlign="Right" />  
<ColumnDirective field='Freight' width='100' format="C2" textAlign="Right"  
/>  
<ColumnDirective field='ShipCountry' width='100' />  
</ColumnsDirective>  
</GridComponent>  
)  
;  
}
```

Run the project

To run the project, use the following command:

```
`bash  
gulp serve  
`
```

The output will appear as follows:

OrderID	CustomerID	EmployeeID	Freight	ShipCountry
10248	VINET	5	\$32.38	France
10249	TOMSP	6	\$11.61	Germany
10250	HANAR	4	\$65.83	Brazil

Getting Started with the Preact Framework with Syncfusion React Components

This article provides a step-by-step guide for setting up a [Preact](#) project and integrating the Syncfusion React components.

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 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 Grid component](#) as an example. To use the React Grid component in the project, the `@syncfusion/ej2-react-grids` package needs to be installed using the following command:

```
`bash
```

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

,

or

```
`bash
```

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

,

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 Grid 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-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-grids/styles/material3.css";
```

The order of importing CSS styles should be in line with its dependency graph.

Add Syncfusion React component

Follow the below steps to add the React Grid component to the Vite project:

1\ Before adding the Grid component to your markup, import the Grid component in the **src/index.jsx** file.

~/SRC/INDEX.JSX

```
import { GridComponent, ColumnsDirective, ColumnDirective } from
'@syncfusion/ej2-react-grids';
```

2\ Then, define the Grid component with the [dataSource](#) property and column definitions. Declare the values for the **dataSource** property.

~/SRC/INDEX.JSX

```
import { render } from 'preact';
import { ColumnDirective, ColumnsDirective, GridComponent } from
'@syncfusion/ej2-react-grids';
import './style.css';
export function App() {
  const data = [
    {
      OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
      Freight: 32.38
    },
    {
      OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
      Freight: 11.61
    },
    {
      OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
      Freight: 65.83
    }
  ];
  return (
    <GridComponent dataSource={data}>
      <ColumnsDirective>
```

```

<ColumnDirective field='OrderID' width='100' textAlign="Right"/>
<ColumnDirective field='CustomerID' width='100' />
<ColumnDirective field='EmployeeID' width='100' textAlign="Right"/>
<ColumnDirective field='Freight' width='100' format="C2" textAlign="Right"/>
<ColumnDirective field='ShipCountry' width='100' />
</ColumnsDirective>
</GridComponent>
);
}
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:

OrderID	CustomerID	EmployeeID	Freight	ShipCountry
10248	VINET	5	\$32.38	France
10249	TOMSP	6	\$11.61	Germany
10250	HANAR	4	\$65.83	Brazil

See also

[Getting Started with the Syncfusion React UI Component](#)

Installation

Installation

The Syncfusion React npm packages can be installed in the application through any of the following steps:

- Installation using npm CLI.
- Installation through package reference in `Package.json` file.

Install by using npm CLI

Syncfusion React (Essential JS 2) packages are published in [npm](#). You can install the necessary packages from npm's install command. For example, React Grid package can be installed using following command.

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

The above command will install the Grid component package and all its dependency packages. You can see the `package.json` file updated with Grid package in dependencies.

Install by using package.json

1. Add the Syncfusion React (Essential JS 2) package references in the dependencies of `~/package.json` file.

```
{
  "dependencies": {
    "@syncfusion/ej2-react-grids": "*",
  }
}
```

The `*` indicates the latest version of npm package.

2. Now, open the command prompt and run the `npm install` command line. By default, npm install will install all modules listed as dependencies in `package.json`.

Refer the [documentation](#) for more details about npm package.json

Npm packages for Syncfusion React UI Components

The Syncfusion React UI components are separately available in individual [npm packages](#). The npm packages are segregated based on the component usage.

Anatomy of npm packages

Syncfusion React UI components are shipped as npm packages. Following table explains the purpose of each file available in the package.

Files	Purpose
<code>dist/es6</code>	This folder contains the ES6 formatted JS file of the package.

| `dist/<packagename>.umd.min.js` `dist/<packagename>.umd.js` | For applications using AMD or Common JS based module loader can be utilize these files.

|

| `src/` | This folder contains the script files in AMD format. You can connect these AMD files as a package in System JS or Require JS.

|

| `styles/<theme name>.css` `styles/<theme name>.scss` | This folder contains the CSS and SCSS files of the package.

||

Available npm package

Syncfusion React Base

The Syncfusion React Base is a common package of Essential JS 2 for React which contains base libraries, methods and class definitions.

Package Name	Installation command
----- -----	
ej2-react-base	npm install @syncfusion/ej2-react-base

Syncfusion React BarcodeGenerator

The Syncfusion Essential JS Barcode widget enables rendering of one dimension and two dimension barcodes in web page. Barcode provides you a simple and inexpensive method of encoding text information that can be easily read by electronic readers.

Package Name	Installation command	Components
----- ----- -----		
ej2-react-barcode-generator	npm i @syncfusion/ej2-react-barcode-generator	BarcodeGenerator

Syncfusion React Buttons

The Syncfusion React buttons package contains UI components such as Button, Checkbox, RadioButton, Switch, and Chip component.

<!-- markdownlint-disable MD033 -->

Package Name	Installation command	Components
----- ----- -----		
ej2-react-buttons	npm install @syncfusion/ej2-react-buttons	• Button • CheckBox • Chip • RadioButton • Switch

Syncfusion React Calendars

The Syncfusion React Calendars package contains date and time components such as Calendar, DatePicker, DateRangePicker, DateTimePicker, and TimePicker. These components come with options to disable dates, restrict selection, and show custom events.

<!-- markdownlint-disable MD033 -->

Package Name	Installation command	Components
----- ----- -----		

| [ej2-react-calendars](#) | npm install @syncfusion/ej2-react-calendars | • Calendar
 • DatePicker
 • DateRangePicker
 • DateTimePicker
 • TimePicker
 |

Syncfusion React Charts

The Syncfusion React Chart control is used to visualize the data with user interactivity and provides customizing options to configure the data visually. It can bind data from datasource such as array of JSON objects , [OData web services](#) or [DataManager](#). All chart elements are rendered using Scalable Vector Graphics (SVG).

<!-- markdownlint-disable MD033 -->

Package Name	Installation command	Components
-----	-----	-----
ej2-react-charts	npm install @syncfusion/ej2-react-charts	• Accumulation Chart • Bullet Chart • Chart • Range Navigator • Smith Chart • Sparkline • Stock Chart

Syncfusion React CircularGauge

The Syncfusion React CircularGauge component is ideal to visualize numeric values over a circular scale. All the circular gauge elements are rendered using Scalable Vector Graphics (SVG).

Package Name	Installation command	Components
-----	-----	-----
ej2-react-circulargauge	npm i @syncfusion/ej2-react-circulargauge	CircularGauge

Syncfusion React Data

It is a data management package to perform data operations such as grouping, sorting in client applications. It will act as an abstraction for using local data sources like array of JavaScript objects and remote data sources like web services returning JSON, JSONP, oData or XML.

Package Name	Installation command
-----	-----
ej2-data	npm i @syncfusion/ej2-data

Syncfusion React Diagrams

The Syncfusion React Diagram component visually represents information. It is also used to create diagrams like flow charts, organizational charts, mind maps, and BPMN either through code or a visual interface.

Package Name	Installation command	Components
-----	-----	-----
ej2-react-diagrams	npm i @syncfusion/ej2-react-diagrams	Diagrams

Syncfusion React DropDowns

The Syncfusion React DropDowns package contains a collection of Dropdown components such as Dropdown List, Combo Box, AutoComplete, Multiselect Dropdown, and List Box. Dropdown components contain specific features such as data binding, grouping, sorting, filtering, and templates.

<!-- markdownlint-disable MD033 -->

Package Name	Installation command	Components
--------------	----------------------	------------

Package Name	Installation command	Components
ej2-react-dropdowns	<code>npm i @syncfusion/ej2-react-dropdowns</code>	AutoComplete ComboBox Dropdown List Dropdown Tree ListBox Multiselect Dropdown

Syncfusion React FileManager

The Syncfusion File Manager is a graphical user interface component used to manage the file system. It enables the user to perform common file operations such as accessing, editing, uploading, downloading, and sorting files and folders. This component also allows easy navigation for browsing or selecting a file or folder from the file system.

Package Name	Installation command	Components
ej2-react-filemanager	<code>npm i @syncfusion/ej2-react-filemanager</code>	FileManager

Syncfusion React Gantt

The Syncfusion React Gantt is designed to visualize and edit the project schedule, and track the project progress. It helps to organize and schedule the projects, and also the project schedule can be updated through interactions like editing, dragging, and resizing.

Package Name	Installation command	Components
ej2-react-gantt	<code>npm i @syncfusion/ej2-react-gantt</code>	Gantt

Syncfusion React Grid

The React Data Grid component is used to display and manipulate tabular data with configuration options to control the way the data is presented and manipulated. It will pull data from a data source, such as array of JSON objects, OData web services, or [DataManager](#) binding data fields to columns. Also displaying a column header to identify the field with support for grouped records.

Package Name	Installation command	Components
ej2-react-grids	<code>npm install @syncfusion/ej2-react-grids</code>	Grid

Syncfusion React HeatMap

The Syncfusion React Heat map is used to visualize a two-dimensional data in which the values are represented in gradient or fixed colors.

Package Name	Installation command	Components
ej2-react-heatmap	<code>npm install @syncfusion/ej2-react-heatmap</code>	HeatMap

Syncfusion React InPlaceEditor

The Syncfusion React InPlace Editor component is most useful for editing a value dynamically within its context (in-place). Its features include inline and pop-up modes, and customizable user interface (UI) and events.

Package Name	Installation command	Components

| [ej2-react-inplace-editor](#) | npm install @syncfusion/ej2-react-inplace-editor | InPlaceEditor |

Syncfusion React Inputs

The Syncfusion React Input components comes with a collection of form components which is useful to get different input values from the users such as text, numbers, patterns, color and file inputs.

<!-- markdownlint-disable MD033 -->

| Package Name | Installation command | Components |

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

| [ej2-react-inputs](#) | npm install @syncfusion/ej2-react-inputs | • ColorPicker
 • MaskedTextBox
 • NumericTextBox
 • Slider
 • Signature
 • Textbox
 • Uploader
 |

Syncfusion React Kanban

The Kanban Board component is an efficient way to visualize workflow at each stage along its path to completion. The control supports necessary features to design task scheduling applications. The key features are swimlanes, customizable cards, binding from local and remote data sources, columns mapping, stacked headers, WIP validation, templating, responsiveness, filtering, and editing.

| Package Name | Installation command | Components |

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

| [ej2-react-kanban](#) | npm install @syncfusion/ej2-react-kanban | Kanban |

Syncfusion React Layouts

The Syncfusion Layout package contains Cards, Avatars, Splitter and Dashboard Layout components.

- The **Card** is a small container in which user can show defined content in specific structure.
- The **Avatars** are icons, initials or figures representing a particular person, used in popular media formats like images, SVG, font icons, and letters.
- The **splitter** is container component which used to construct different layouts using multiple and nested panes.
- The **Dashboard Layout** is a grid structured layout component that helps to create a dashboard with panels. Panels hold the UI components and allow resize, reorder, drag-n-drop, remove and add options. This allows users to easily place the components at the desired position within the grid layout.

<!-- markdownlint-disable MD033 -->

| Package Name | Installation command | Components |

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

| [ej2-react-layouts](#) | npm install @syncfusion/ej2-react-layouts | • Avatar
 • Card
 • Dashboard Layout
 • Splitter
 |

Syncfusion React LinearGauge

The Syncfusion Linear Gauge is used for visualizing numeric values in a linear scale with features like multiple axes, different orientations, and more. The appearance of the gauge can be completely customized to simulate a thermometer, pressure gauge, ruler, etc.

| Package Name | Installation command | Components |

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

| [ej2-react-lineargauge](#) | npm install @syncfusion/ej2-react-lineargauge | LinearGauge |

Syncfusion React Lists

The Syncfusion React ListView component allows to select an item or multiple items from a list-like interface and represents the data in an interactive hierarchical structure across different layouts or views. Lists are used for displaying data, data navigation, and data entry.

|Package Name |Installation command|Components |

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

| [ej2-react-lists](#) | npm install @syncfusion/ej2-react-lists | ListView |

Syncfusion React Maps

The Syncfusion React Maps is used to visualize the geographical data and represent the statistical data of a particular geographical area on earth with user interactivity, and provides various customizing options.

|Package Name |Installation command|Components |

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

| [ej2-react-maps](#) | npm install @syncfusion/ej2-react-maps | Maps |

Syncfusion React Navigations

A package of React navigation components such as Accordion, Breadcrumb, ContextMenu, MenuBar, Tabs, Toolbar, TreeView, and Sidebar.

<!-- markdownlint-disable MD033 -->

|Package Name |Installation command|Components |

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

| [ej2-react-navigations](#) | npm install @syncfusion/ej2-react-navigations | • Accordion
• Breadcrumb
 • ContextMenu
 • MenuBar
 • Tabs
 • Toolbar
 • TreeView
 • Sidebar
 |

Syncfusion React Notifications

The Syncfusion React Notification component is used to notify status or summary information to the end-users.

<!-- markdownlint-disable MD033 -->

|Package Name |Installation command|Components |

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

| [ej2-react-notifications](#) | npm install @syncfusion/ej2-react-notifications | • Badge
• Spinner
 • Toast
 |

Syncfusion React PdfViewer

The Syncfusion React PDF Viewer supports viewing and reviewing PDF files in web applications and also printing them. The thumbnail, bookmark, hyperlink, and table of contents supports provide easy navigation within and outside the PDF files. The form-filling support provides a platform to fill and print with AcroForms. The PDF files can be reviewed with the available annotation tools.

Package Name	Installation command	Components
ej2-react-pdfviewer	<code>npm install @syncfusion/ej2-react-pdfviewer</code>	PdfViewer

Syncfusion React PivotTable

The Syncfusion React Pivot Table is a powerful control used to organize and summarize business data and display the result in a cross-table format. It includes major functionalities such as data binding, drilling up and down, Excel-like filtering and sorting, editing, Excel and PDF exporting, several built-in aggregations, pivot table field list, and calculated fields.

Package Name	Installation command	Components
ej2-react-pivotview	<code>npm install @syncfusion/ej2-react-pivotview</code>	PivotTable

Syncfusion React Popups

A package of Popup components Dialog and Tooltip are used to display information or to get input from the users in a popup.

<!-- markdownlint-disable MD033 -->

Package Name	Installation command	Components
ej2-react-popups	<code>npm install @syncfusion/ej2-react-popups</code>	Dialog Tooltip

Syncfusion React ProgressBar

The Progress Bar can be used to visualize the changing status of an extended operation such as a download, file transfer, or installation. All the progress bar elements are rendered using scalable vector graphics (SVG) to ensure the quality of the visual experience.

Package Name	Installation command	Components
ej2-react-progressbar	<code>npm install @syncfusion/ej2-react-progressbar</code>	ProgressBar

Syncfusion React QueryBuilder

The Syncfusion React QueryBuilder package contains the QueryBuilder component that allows the users to create and edit filters. It supports data binding, templates, validation, and horizontal and vertical orientation.

Package Name	Installation command	Components
ej2-react-querybuilder	<code>npm install @syncfusion/ej2-react-querybuilder</code>	QueryBuilder

Syncfusion React RichTextEditor

The RichTextEditor component is the HTML and markdown editor that provides the best user experience for creating, updating, and formatting the content.

Package Name	Installation command	Components

| [ej2-react-richtexteditor](#) | npm install @syncfusion/ej2-react-richtexteditor | RichTextEditor |

Syncfusion React Schedule

The Syncfusion React Scheduler component is an event calendar that facilitates users with the common Outlook-calendar features, thus allowing them to plan and manage their events/appointments and their time in an efficient way.

| Package Name | Installation command | Components |

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

| [ej2-react-schedule](#) | npm install @syncfusion/ej2-react-schedule | Schedule |

Syncfusion React SplitButtons

The Syncfusion React SplitButtons package contains UI components such as DropDownButton, SplitButton, ProgressButton, and ButtonGroup components. DropDownButton and SplitButton component display a list of items when a button is clicked and the ButtonGroup can be used for easy navigation.

<!-- markdownlint-disable MD033 -->

| Package Name | Installation command | Components |

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

| [ej2-react-splitbuttons](#) | npm install @syncfusion/ej2-react-splitbuttons | • ButtonGroup
• DropDownButton
• ProgressButton
• SplitButton
 |

Syncfusion React Spreadsheet

The Syncfusion React Spreadsheet is an user interactive component to organize and analyze data in tabular format with configuration options for customization. It will load data by importing an excel file or from a data source, such as RESTful JSON data services and local JavaScript object array binding. The populated data can be exported as Excel with xlsx, xls and csv formats.

| Package Name | Installation command | Components |

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

| [ej2-react-spreadsheet](#) | npm install @syncfusion/ej2-react-spreadsheet | Spreadsheet |

Syncfusion React TreeGrid

The Syncfusion React Tree Grid is a feature-rich control used to visualize self-referential and hierarchical data effectively in a tabular format. It can pull data from data sources such as an enumerable collection of records, RESTful services, OData services, WCF services, or DataManager. It also expands or collapses child data using the tree column.

| Package Name | Installation command | Components |

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

| [ej2-react-treegrid](#) | npm install @syncfusion/ej2-react-treegrid | TreeGrid |

Syncfusion React TreeMap

The Syncfusion React TreeMap is a feature-rich component used to visualize both hierarchical and flat data. The look and feel of the treemaps can be customized by using the built-in features like color mapping, legends, and label templates.

| Package Name | Installation command | Components |

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

| [ej2-react-treemap](#) | npm install @syncfusion/ej2-react-treemap | TreeMap |

Syncfusion React WordProcessor

The Syncfusion React Word Processor (Document Editor) is a component with editing capabilities like Microsoft Word. It is used to create, edit, view, and print Word documents. It provides all the common Word processing features including editing text, formatting contents, resizing images and tables, finding and replacing text, bookmarks, tables of contents, printing, and importing and exporting Word documents.

|Package Name |Installation command|Components |

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

| [ej2-react-documenteditor](#) | npm install @syncfusion/ej2-react-documenteditor | DocumentEditor |

See also

- [Installation with npm CLI](#)
- [Download JavaScript – EJ2 Installer](#)
- [Product Development Life Cycle](#)
- [Update npm Packages](#)

Web Installer

Download JavaScript – EJ2 Installer

The Syncfusion JavaScript - EJ2 web installer can be downloaded from the [Syncfusion](#) website. You can either download the licensed installer or try our trial installer depending on your license.

- Trial Installer
- Licensed Installer

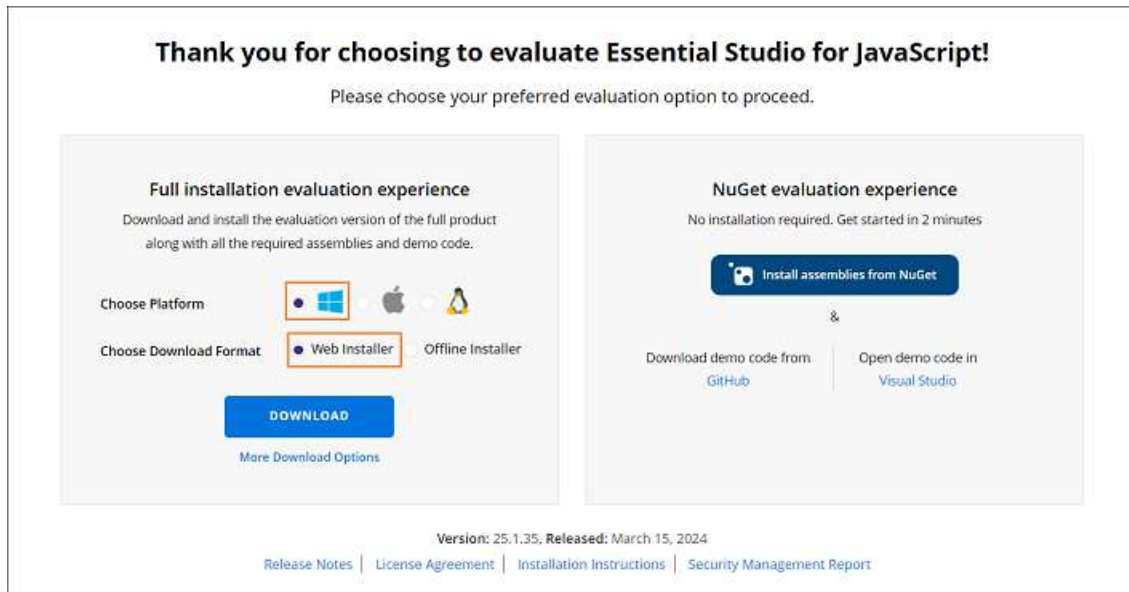
Download the free trial version

Our 30-day trial can be downloaded in two ways.

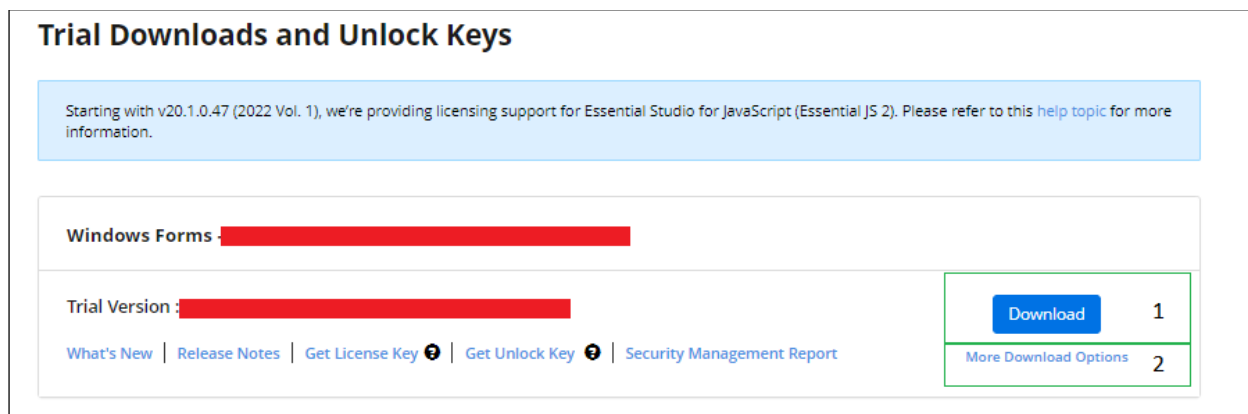
- Download Free Trial Setup
- Start Trials if using components through [npm](#)

Download free trial setup

1. You can evaluate our 30-day free trial by visiting the [Download Free Trial](#) page and select the JavaScript platform.
2. After completing the required form or logging in with your registered Syncfusion account, you can download the JavaScript - EJ2 trial installer from the confirmation page. (See the screenshot below.)



3. With a trial license, only the latest version's trial installer can be downloaded.
4. After downloading, the Syncfusion JavaScript - EJ2 trial installer can be unlocked using either the trial unlock key or the Syncfusion registered login credential.
5. Before the trial expires, you can download the trial installer at any time from your registered account's [Trials & Downloads](#) page (See the screenshot below.)
6. Click the Download (element 1 in the screenshot below) button to get the Syncfusion Essential Studio JavaScript – EJ2 web installer.



Start Trials if using components through [npm](#)

You should initiate an evaluation if you have already obtained our components through [npm](#)

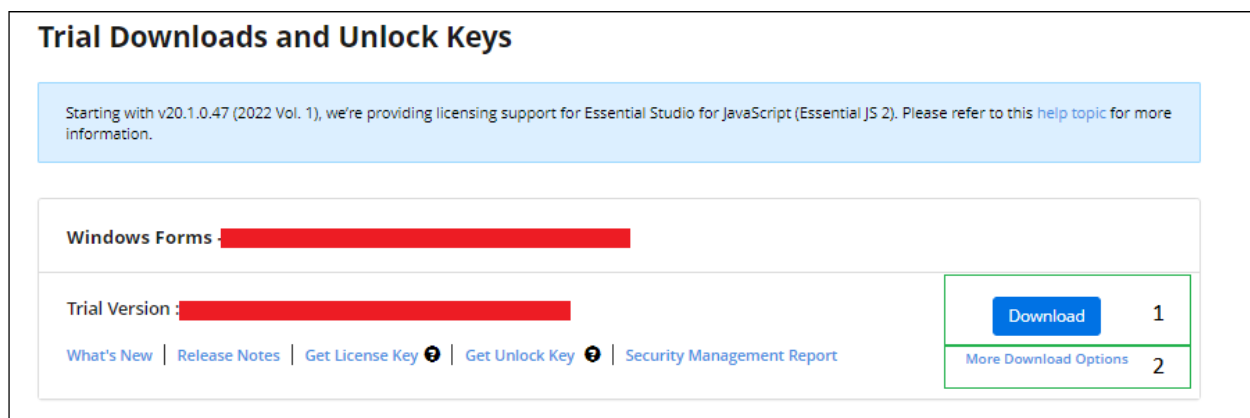
1. You can start your 30-day free trial for JavaScript – EJ2 from the [Start Trial](#) page from your account.



2. To access this page, you must sign up\log in with your Syncfusion account.
3. Begin your trial by selecting the JavaScript – EJ2 product.

Note: If you've already used the trial products and they haven't expired, you won't be able to start the trial for the same product again.

4. After you've started the trial, go to the [Trials & Downloads](#) page to get the latest version trial installer. You can generate the [unlock](#) key and [license key](#) here at any time before the trial period expires. (See the screenshot below). You can find your current active trial products on the [Trials & Downloads](#) page.



5. You can find your current active trial products on the [Trials & Downloads](#) page.

Download the license version

1. Syncfusion licensed products will be available in the [License & Downloads](#) page under your registered Syncfusion account.
2. You can view all the licenses (both active and expired) associated with your account.
3. Click the **Download** (element 1 in the screenshot below) button to download the respective product's installer.
4. The most recent version of the installer will be downloaded from this page.
5. To download older version installers, go to [Downloads Older Versions](#) (element 2 in the screenshot below).
6. You can download other platform\add-on installers by going to **More Downloads Options** (element 3 in the screenshot below).



You can refer to the [Online installer](#) link for step-by-step installation guidelines.

Installation using Web Installer

Overview

For the Essential Studio JavaScript – EJ2 product, Syncfusion offers a Web Installer. This installer alleviates the burden of downloading a larger installer. You can simply download and run the online installer, which will be smaller in size and will download and install the Essential Studio products you have chosen. You can get the most recent version of Essential Studio Web Installer [here](#).

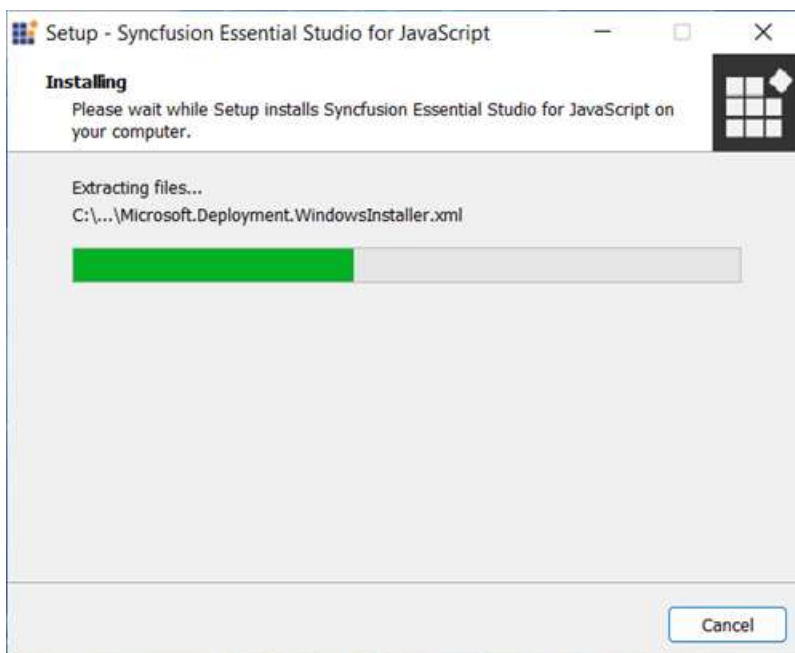
The frameworks listed below are supported in this installer.

- JavaScript
- Angular
- React
- Vue
- JavaScript (ES5)

Installation

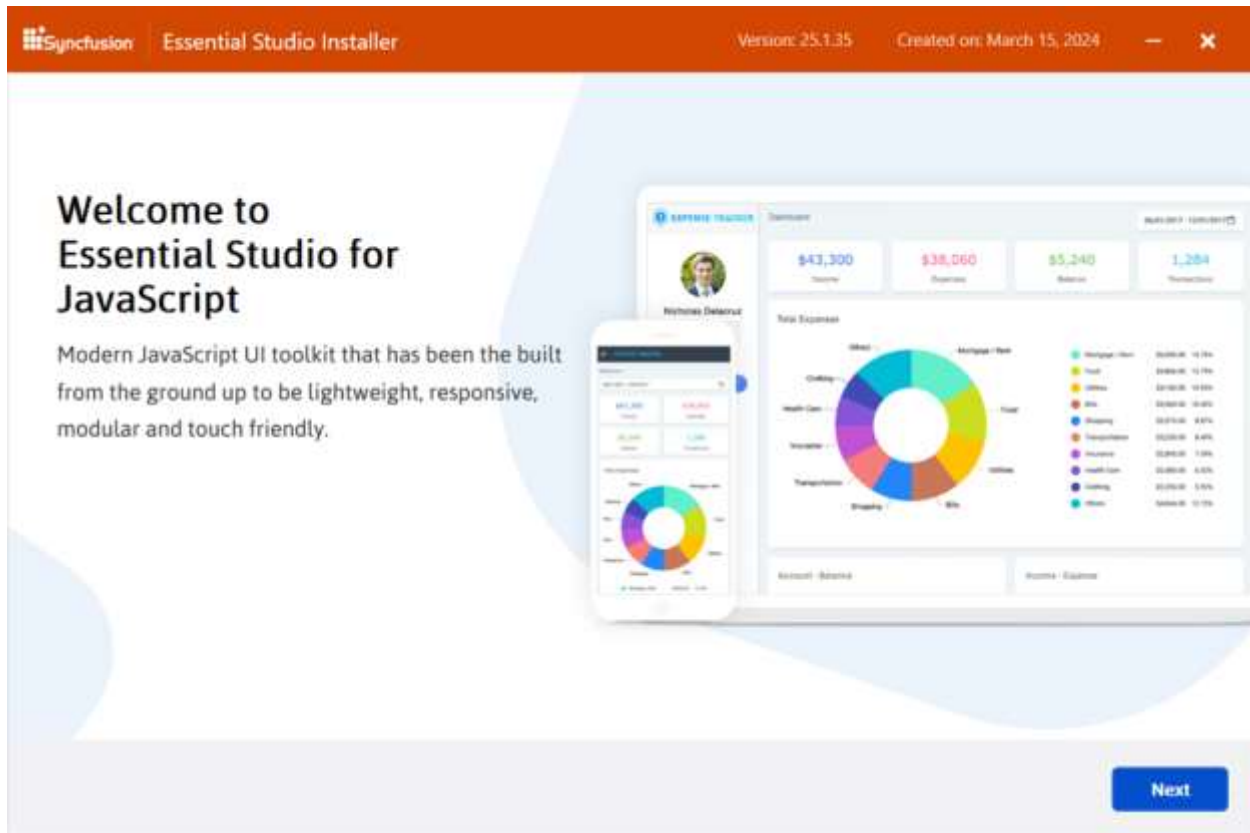
The steps below show how to install Essential Studio JavaScript – EJ2 Web Installer.

1. Open the Syncfusion Essential Studio JavaScript – EJ2 Web Installer file from downloaded location by double-clicking it. The Installer Wizard automatically opens and extracts the package.



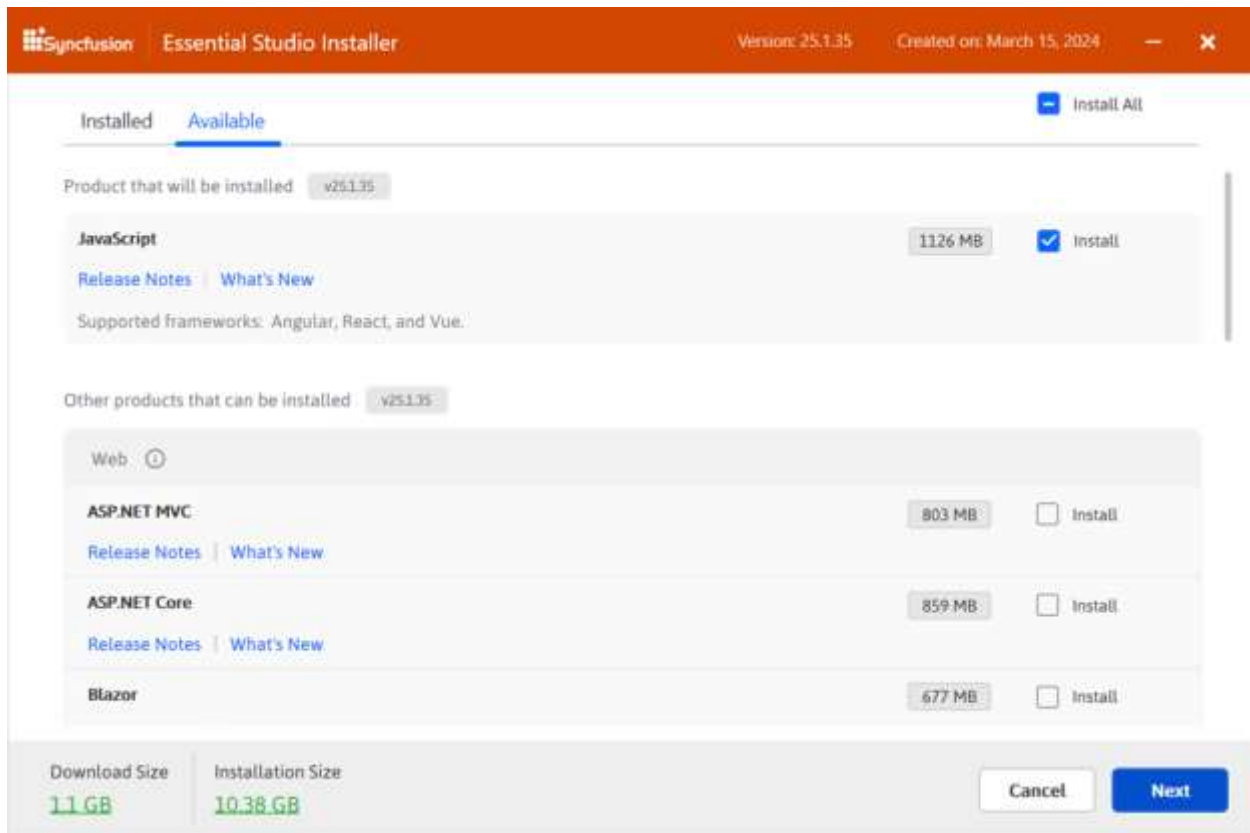
Note: The installer wizard extracts the syncfusionejs2webinstaller_{version}.exe dialog, which displays the package's unzip operation.

2. The Syncfusion JavaScript - EJ2 Web Installer's welcome wizard will be displayed. Click the Next button.



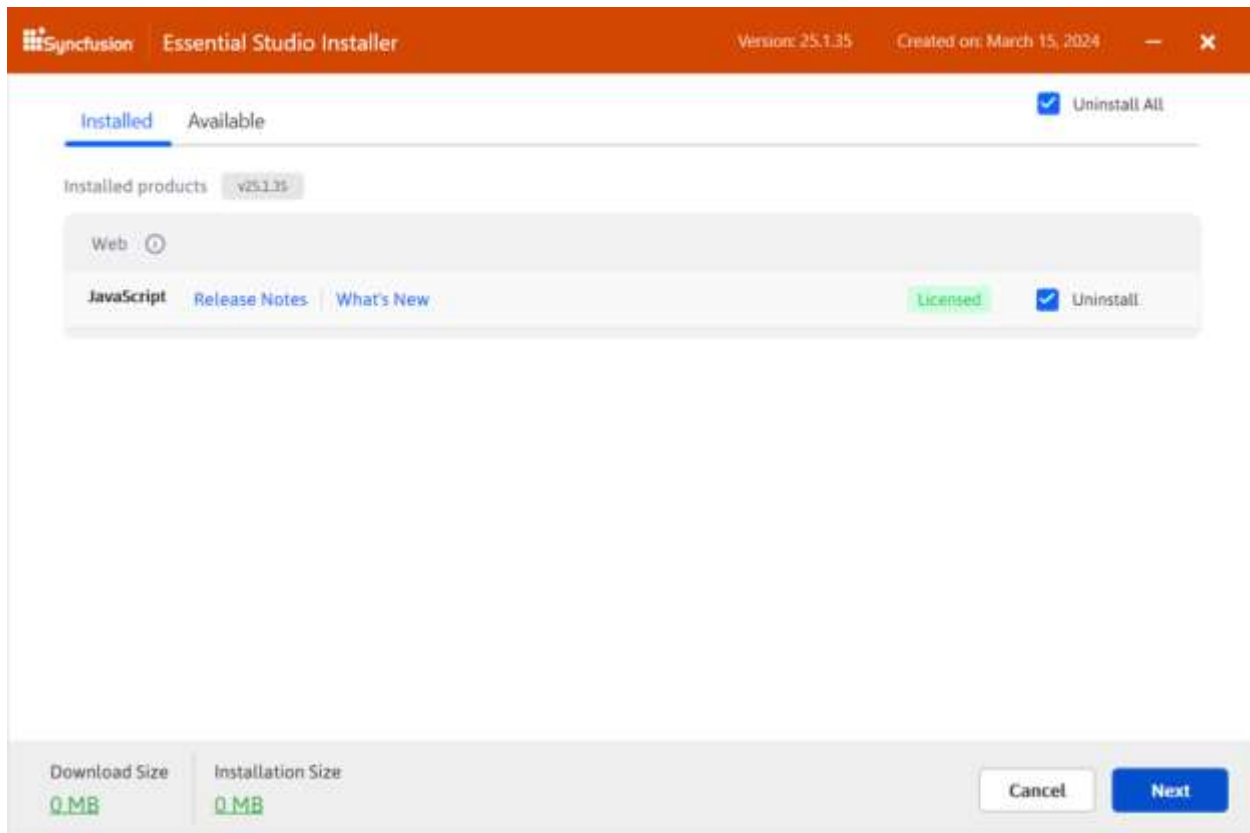
3. The Platform Selection Wizard will appear. From the **Available** tab, select the products to be installed. Select the **Install All** checkbox to install all products.

Available



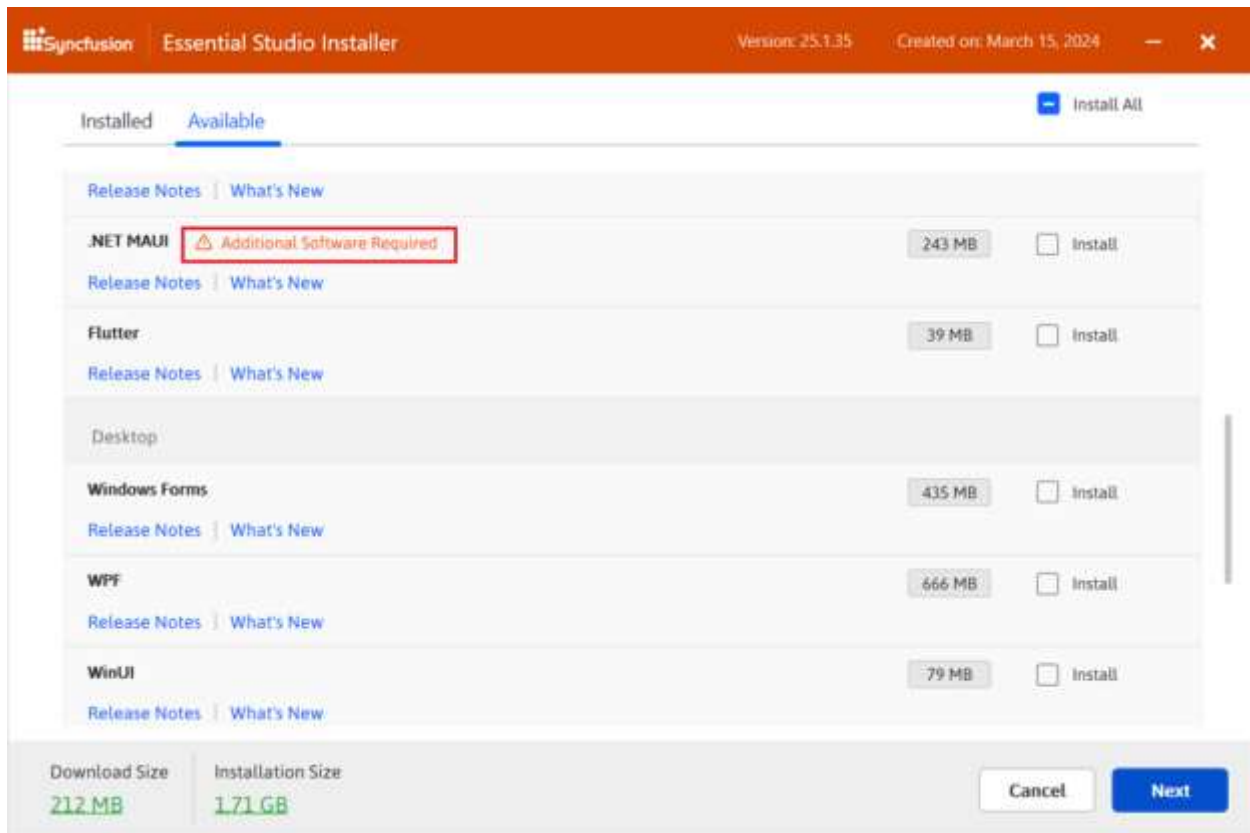
If you have multiple products installed in the same version, they will be listed under the **Installed** tab. You can also select which products to uninstall from the same version. Click the Next button.

Installed

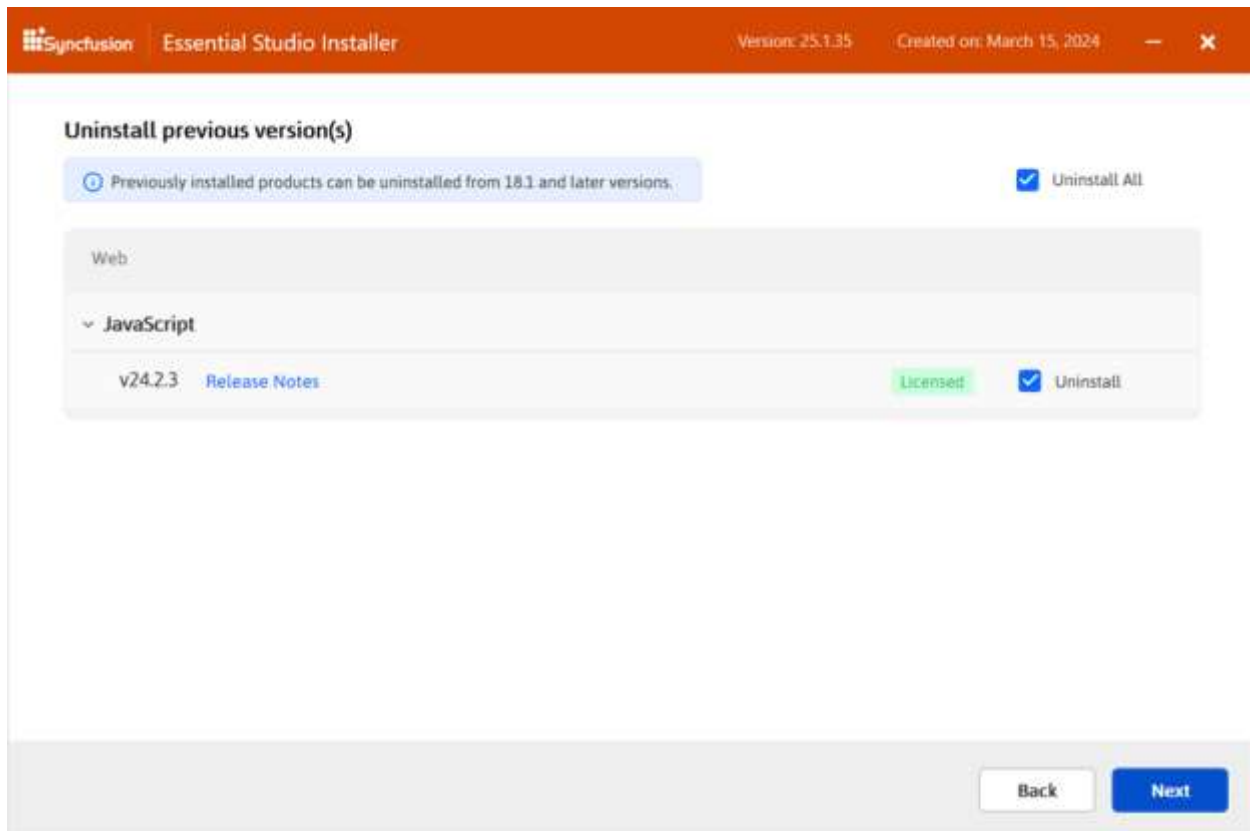


Important: If the required software for the selected product isn't already installed, the **Additional Software Required** alert will appear. You can, however, continue the installation and install the necessary software later.

Required Software

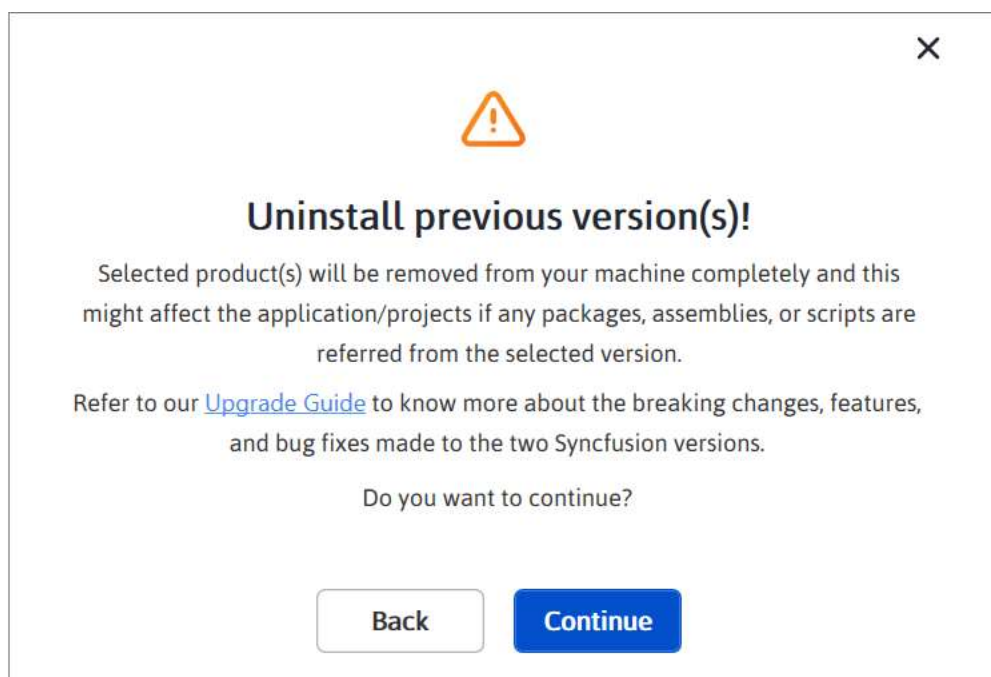


4. If previous version(s) for the selected products are installed, the Uninstall previous version wizard will be displayed. You can see the list of previously installed versions for the products you've chosen here. To remove all versions, check the **Uninstall All** checkbox. Click the Next button.

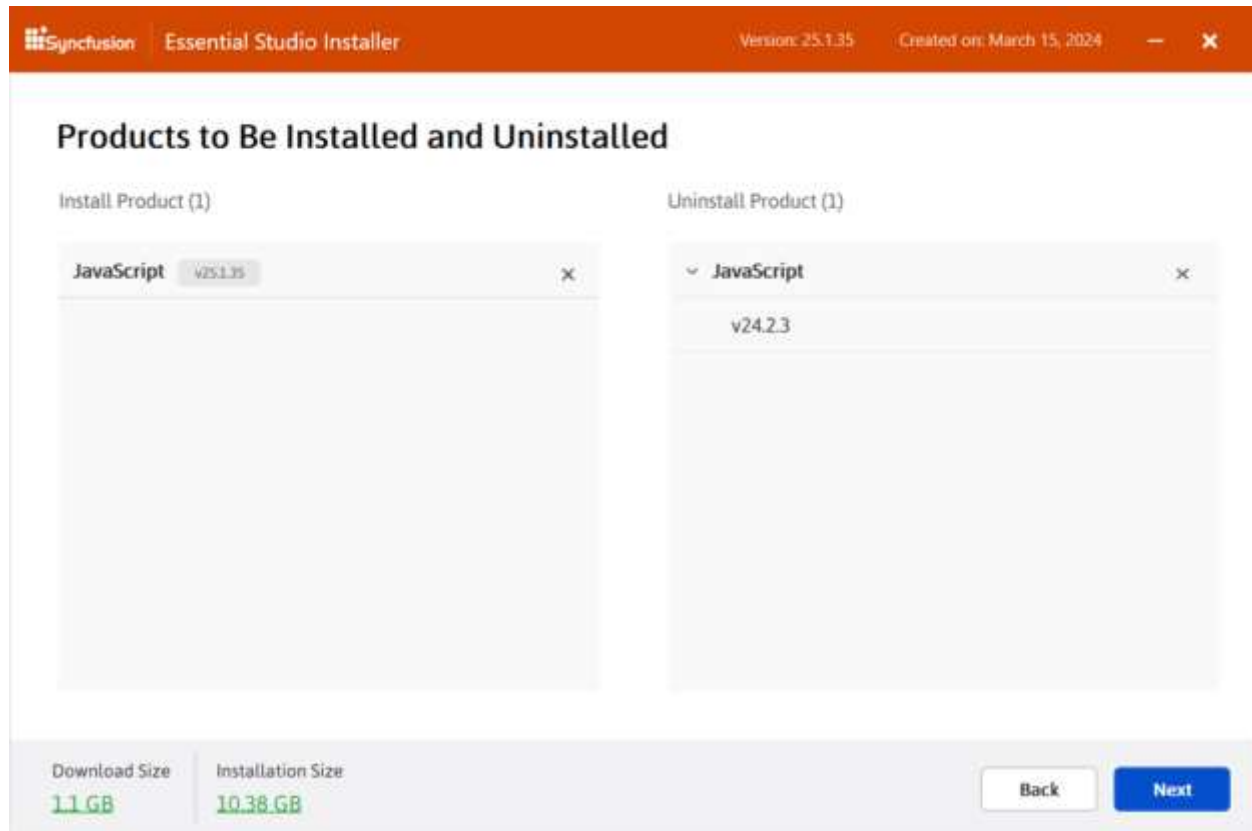


Note: From the 2021 Volume 1 release, Syncfusion has provided option to uninstall the previous versions from 18.1 while installing the new version.

5. Pop up screen will be displayed to get the confirmation to uninstall selected previous versions.



6. The Confirmation Wizard will appear with the list of products to be installed/uninstalled. You can view and modify the list of products that will be installed and uninstalled from this page.



Note: By clicking the **Download Size** and **Installation Size** links, you can determine the approximate size of the download and installation

7. The Configuration Wizard will appear. You can change the Download, Install, and Demos locations from here. You can also change the Additional settings on a product-by-product basis. Click Next to install with the default settings.

Configuration

Download Location
 C:\ProgramData\Syncfusion\25.1.35\Downloads\ [Browse](#)

Installation Location
 C:\Program Files (x86)\Syncfusion\Essential Studio\ [Browse](#)

Demos Location
 C:\Users\Public\Documents\Syncfusion\ [Browse](#)

☒ I Agree to the [License Terms](#) and [Privacy Policy](#)

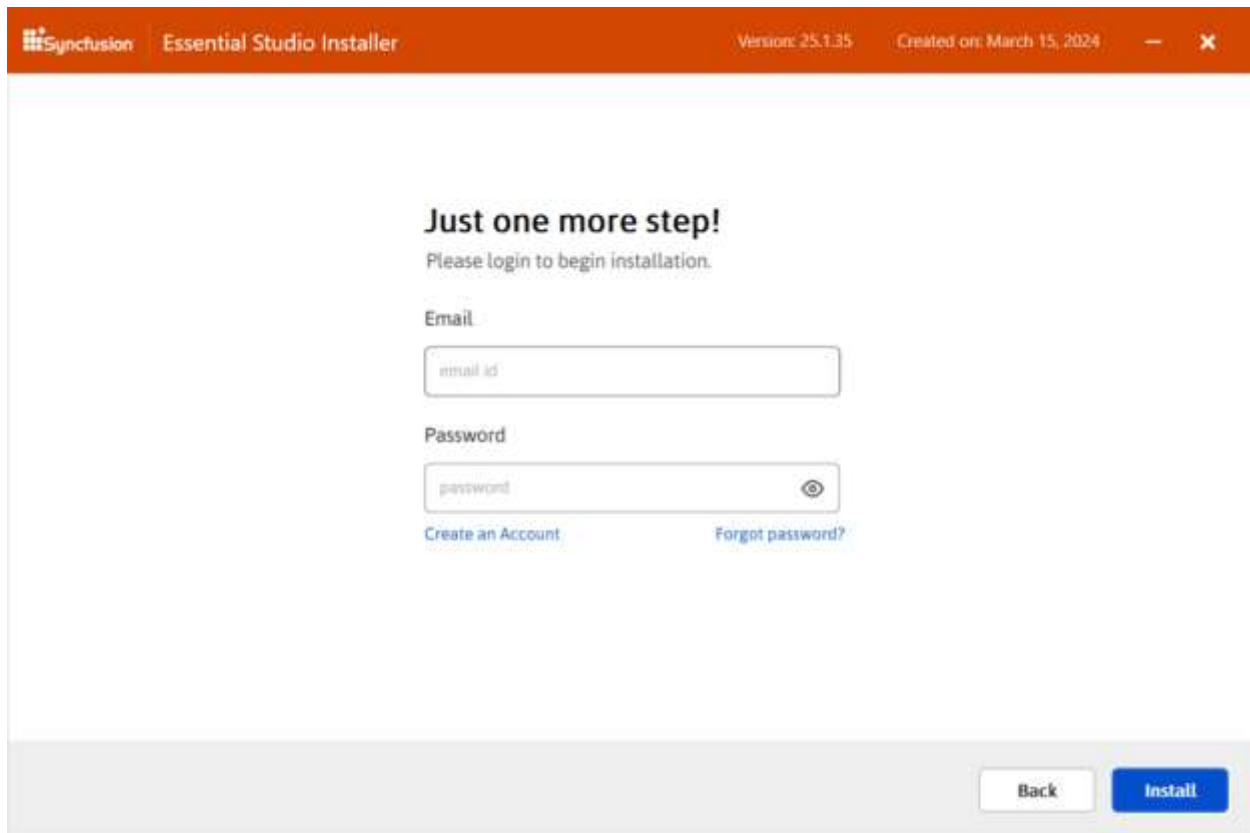
Additional Settings

- ☒ Install Demos
 - ☒ JavaScript
- ☒ Configure Syncfusion Extensions in Visual Studio
 - ☒ JavaScript
- ☒ Create Desktop Shortcut
- ☒ Create Start Menu Shortcut

Download Size: **1.1 GB** Installation Size: **10.38 GB** [Back](#) [Next](#)

Additional settings

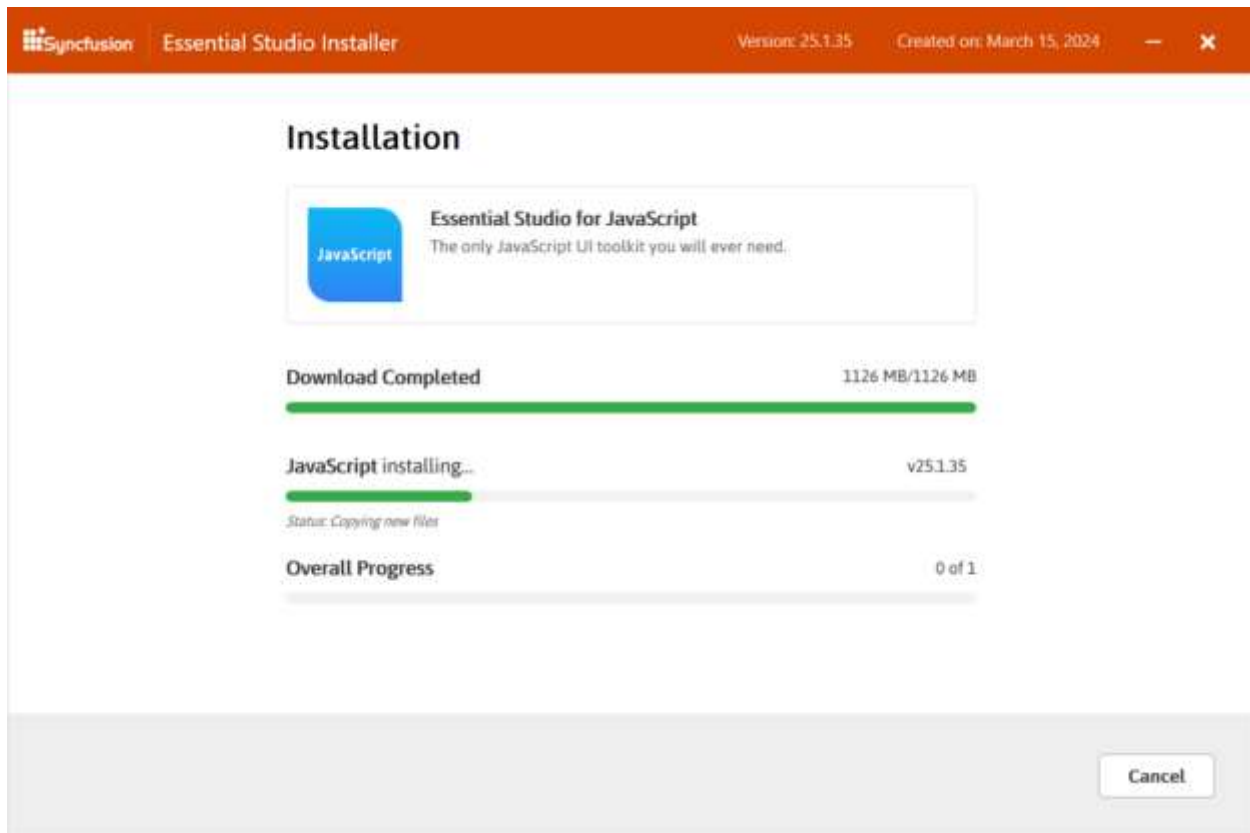
- Select the **Install Demos** check box to install Syncfusion samples, or leave the check box unchecked, if you do not want to install Syncfusion samples
 - Select the **Configure Syncfusion Extensions controls in Visual Studio** checkbox to configure the Syncfusion Extensions in Visual Studio or clear this check box when you do not want to configure the Syncfusion Extensions in Visual Studio.
 - Check the **Create Desktop Shortcut** checkbox to add a desktop shortcut for Syncfusion Control Panel
 - Check the **Create Start Menu Shortcut** checkbox to add a shortcut to the start menu for Syncfusion Control Panel
8. After reading the License Terms and Conditions, check the **I agree to the License Terms and Privacy Policy** check box. Click the Next button.
 9. The login wizard will appear. You must enter your Syncfusion email address and password. If you don't have a Syncfusion account, you can create one by clicking on **Create an Account**. If you have forgotten your password, click **Forgot Password** to create a new one. Click the Install button.



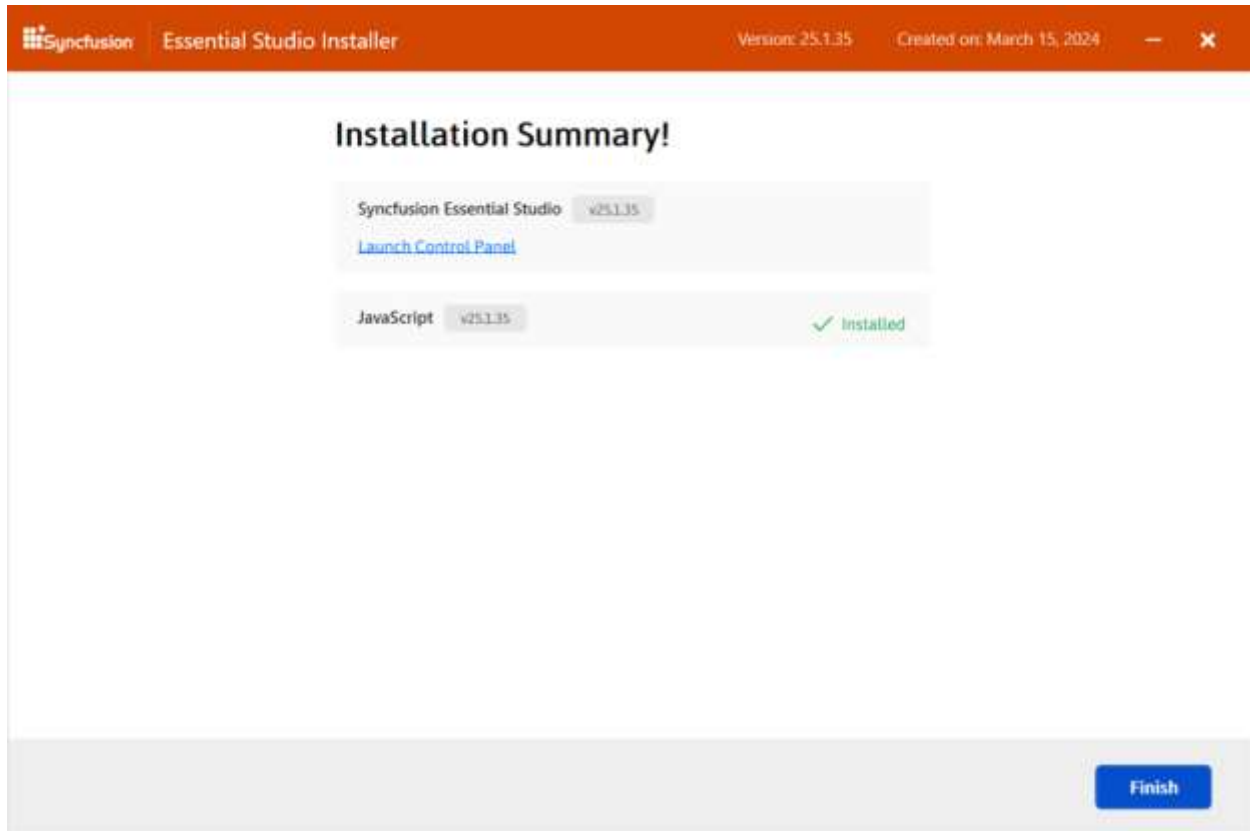
The screenshot shows the 'Essential Studio Installer' window. The title bar is orange and contains the Syncfusion logo, the text 'Essential Studio Installer', and version/creation information: 'Version: 25.1.35' and 'Created on: March 15, 2024'. The main content area is white and features the heading 'Just one more step!' followed by the instruction 'Please login to begin installation.' Below this are two input fields: 'Email' with a placeholder 'email id' and 'Password' with a placeholder 'password' and a toggle icon. At the bottom of the form are two links: 'Create an Account' and 'Forgot password?'. At the bottom right of the window are two buttons: 'Back' and 'Install'.

Important: The products you have chosen will be installed based on your Syncfusion License (Trial or Licensed).

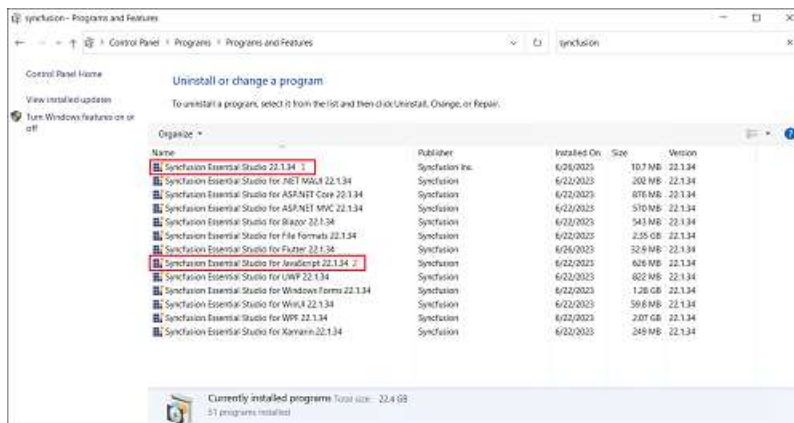
10. The download and installation\uninstallation progress will be displayed as shown below.



11. When the installation is finished, the **Summary** wizard will appear. Here you can see the list of products that have been installed successfully and those that have failed. To close the Summary wizard, click Finish.



- To open the Syncfusion Control Panel, click **Launch Control** Panel.
12. After installation, there will be two Syncfusion control panel entries, as shown below. The Essential Studio entry will manage all Syncfusion products installed in the same version, while the Product entry will only uninstall the specific product setup.



Uninstallation

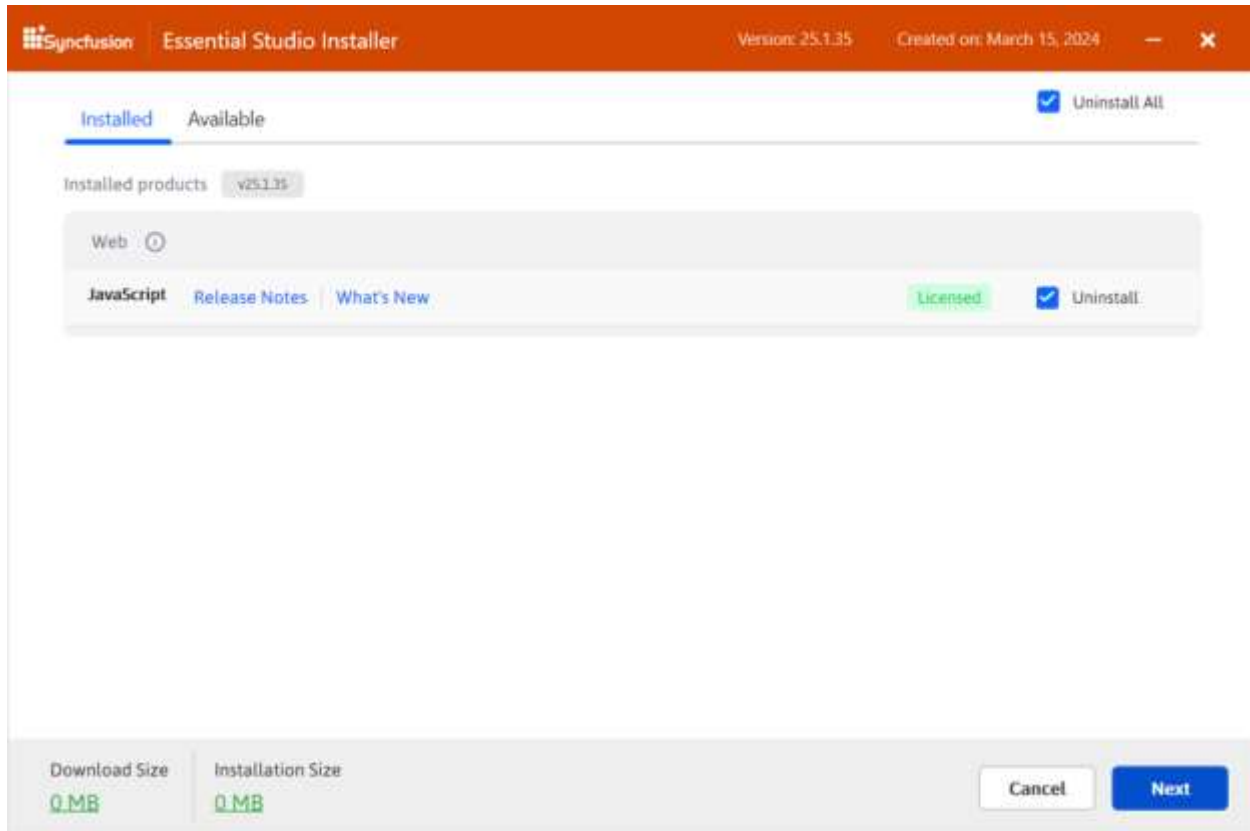
Syncfusion JavaScript – EJ2 installer can be uninstalled in two ways.

- Uninstall the JavaScript – EJ2 using the Syncfusion JavaScript – EJ2 web installer
- Uninstall the JavaScript – EJ2 from Windows Control Panel

Follow either one of the option below to uninstall Syncfusion Essential Studio JavaScript – EJ2 installer

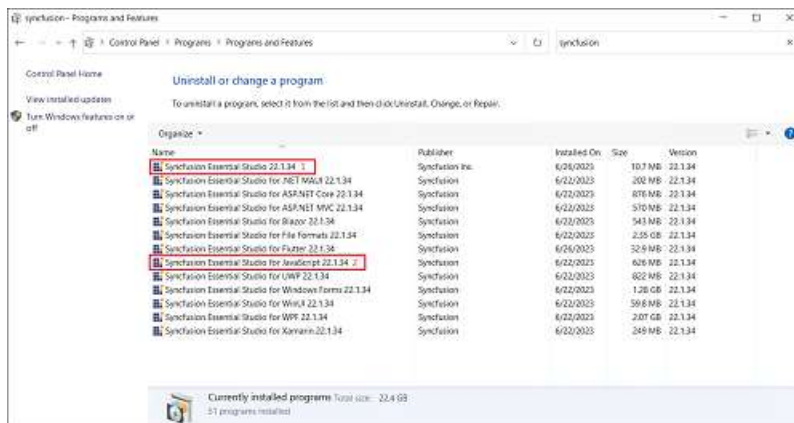
[Option 1: Uninstall the JavaScript–EJ2 using the Syncfusion JavaScript–EJ2 web installer](#)

Syncfusion provides the option to uninstall products of the same version directly from the Web Installer application. Select the products to be uninstalled from the list, and Web Installer will uninstall them one by one.



[Option 2: Uninstall the JavaScript–EJ2 from Windows control panel](#)

You can uninstall all the installed products by selecting the **Syncfusion Essential Studio {version}** entry (element 1 in the below screenshot) from the Windows control panel, or you can uninstall JavaScript – EJ2 alone by selecting the **Syncfusion Essential Studio for JavaScript – EJ2 {version}** entry (element 2 in the below screenshot) from the Windows control panel.



Note: If the **Syncfusion Essential Studio** for JavaScript **{version}** entry is selected from the Windows control panel, the Syncfusion Essential Studio JavaScript – EJ2 alone will be removed and the below default MSI uninstallation window will be displayed.

Offline Installer

Download JavaScript – EJ2 Installer

The Syncfusion JavaScript - EJ2 installer can be downloaded from the Syncfusion website. You can either download the licensed installer or try our trial installer depending on your license.

- Trial Installer
- Licensed Installer

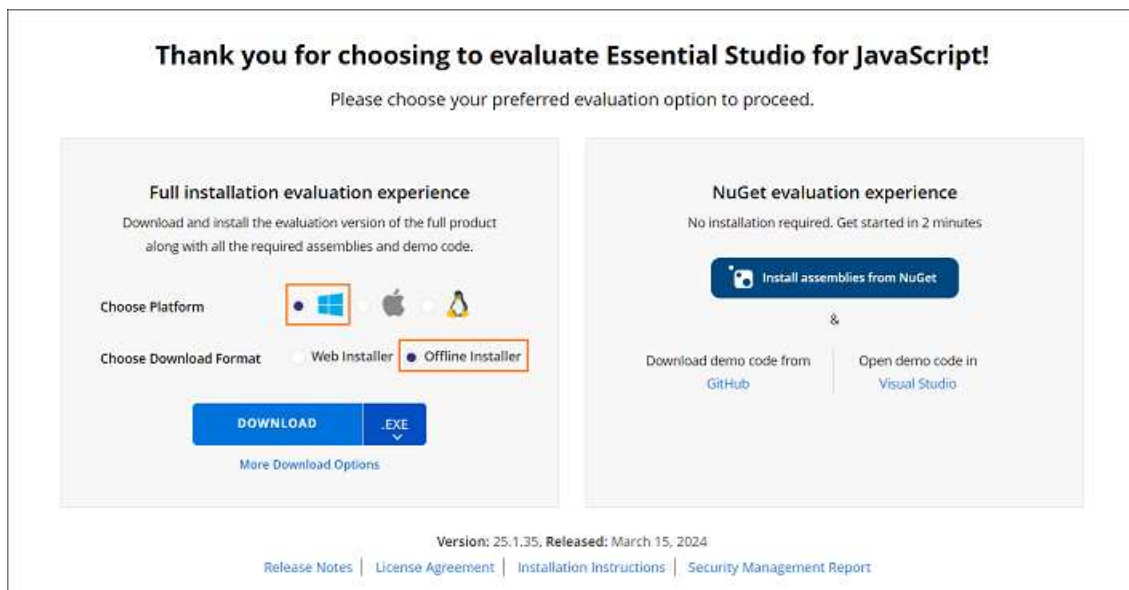
Download the trial version

Our 30-day trial can be downloaded in two ways.

- Download Free Trial Setup
- Start Trials if using components through [npm](#)

Download free trial setup

1. You can evaluate our 30-day free trial by visiting the [Download Free Trial](#) page and select the JavaScript platform.
2. After completing the required form or logging in with your registered Syncfusion account, you can download the JavaScript - EJ2 trial installer from the confirmation page. (See the screenshot below.)



3. With a trial license, only the latest version's trial installer can be downloaded.
4. After downloading, the Syncfusion JavaScript - EJ2 trial installer can be unlocked using either the trial unlock key or the Syncfusion registered login credential. More information on generating an unlock key can be found in [this](#) article.

- Before the trial expires, you can download the trial installer at any time from your registered account's Trials & Downloads page (See the screenshot below.)
- Click the Download (element 1 in the screenshot below) button to get the Syncfusion Essential Studio JavaScript – EJ2 web installer.

Trial Downloads and Unlock Keys

Starting with v20.1.0.47 (2022 Vol. 1), we're providing licensing support for Essential Studio for JavaScript (Essential JS 2). Please refer to this [help topic](#) for more information.

Windows Forms : [Redacted]

Trial Version : [Redacted]

[What's New](#) | [Release Notes](#) | [Get License Key](#) | [Get Unlock Key](#) | [Security Management Report](#)

[Download](#) 1

[More Download Options](#) 2

- Click the More Download Options (element 2 in the above screenshot) button to get the Essential Studio JavaScript – EJ2 Offline trial installer which is available in EXE and ZIP format.

Downloads

Version : 25.1.35
Released on : Mar 15, 2024

[Read our Security Management Report](#) to learn about our security practices.

Windows Mac Linux

WEB - Essential JS 2 Offline Installer Web Installer

JavaScript - EJ 2
Includes Angular, React, and Vue components.
[Release Notes](#) | [Readme](#)

.EXE 1420 MB
Checksum Value

.ZIP 1420 MB
Checksum Value

.EXE 4 MB
Checksum Value

Start Trials if using components through [npm](#)

You should initiate an evaluation if you have already obtained our components through [npm](#)

- You can start your 30-day free trial for JavaScript – EJ2 from the [Start Trial](#) page from your account.

WEB

Blazor
Includes 80+ Blazor components for ASP.NET Core application development.

TRIAL IN PROGRESS

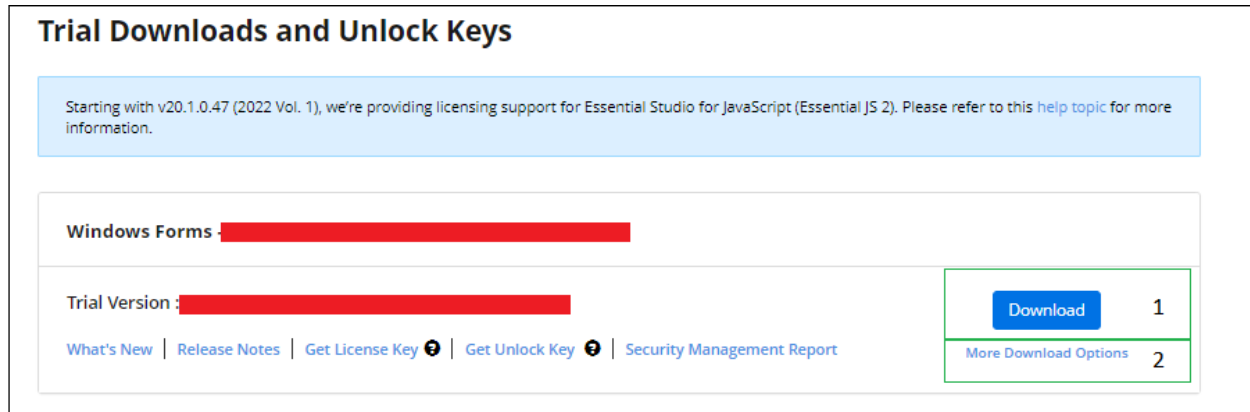
JavaScript
Includes 80+ JavaScript controls for developing modern web applications. It also includes complete support for Angular, React, and Vue frameworks.

START TRIAL

- To access this page, you must sign up\log in with your Syncfusion account.
- Begin your trial by selecting the JavaScript – EJ2 product.

Note: If you've already used the trial products and they haven't expired, you won't be able to start the trial for the same product again.

- After you've started the trial, go to the [Trials & Downloads](#) page to get the latest version trial installer. You can generate the [unlock](#) key here at any time before the trial period expires. (See the screenshot below.)



- You can find your current active trial products on the [Trials & Downloads](#) page.

Download the license version

- Syncfusion licensed products will be available in the [License & Downloads](#) page under your registered Syncfusion account.
- You can view all the licenses (both active and expired) associated with your account.
- Click the Download (element 1 in the screenshot below) button to download the respective product's installer.
- The most recent version of the installer will be downloaded from this page.
- To download older version installers, go to [Downloads Older Versions](#) (element 2 in the screenshot below).
- You can download other platform\add-on installers by going to More Downloads Options (element 3 in the screenshot below).
- For Windows OS, EXE and Zip formats are available for download. They are both Offline Installers.



You can also refer to the [Offline installer](#) link for step-by-step installation guidelines.

Installation using Offline Installer

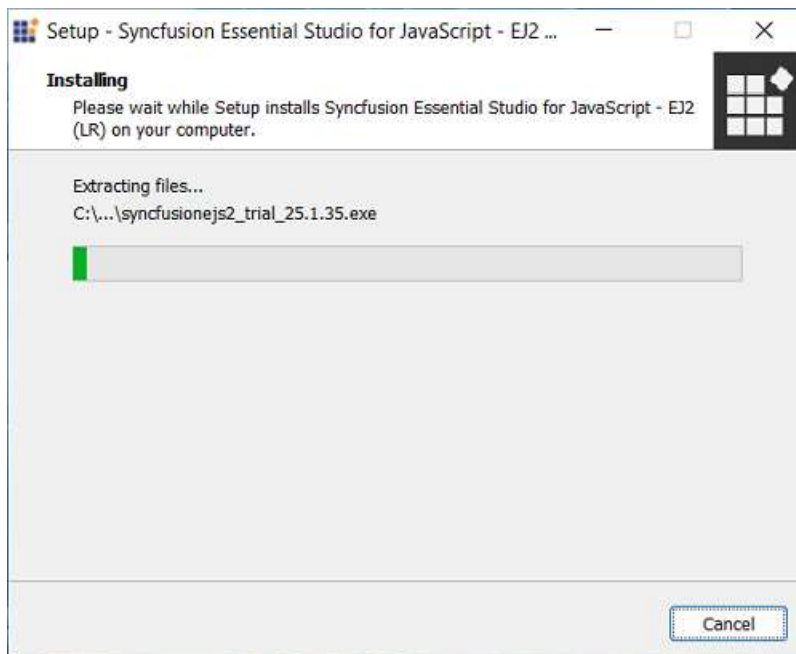
The frameworks listed below are supported in this installer.

- JavaScript
- Angular
- React
- Vue
- JavaScript (ES5)

Installing with UI

The steps below shows how to install the Essential Studio JavaScript – EJ2 installer.

1. Open the Syncfusion JavaScript – EJ2 offline installer file from downloaded location by double-clicking it. The Installer Wizard automatically opens and extracts the package

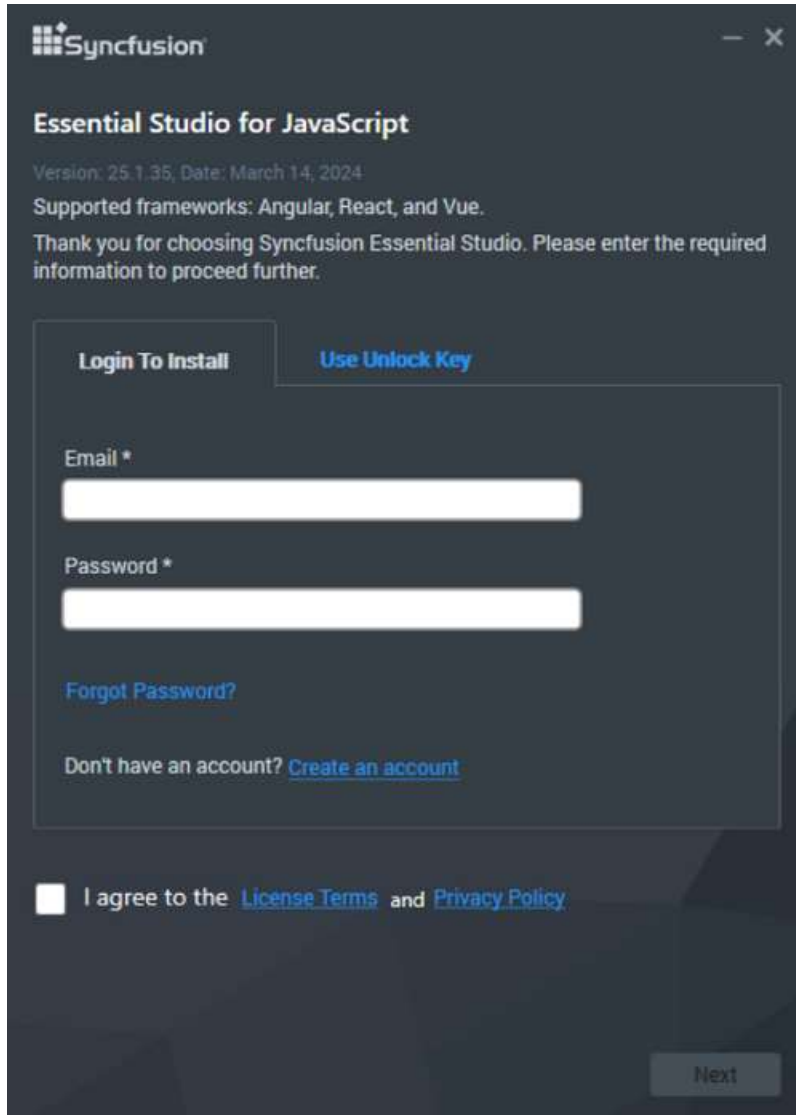


Note: The Installer wizard extracts the syncfusionejs2 (version).exe dialog, which displays the package's unzip operation.

2. To unlock the Syncfusion offline installer, you have two options:
 - Login To Install
 - Use Unlock Key

Login To Install

You must enter your Syncfusion email address and password. If you don't have a Syncfusion account, you can sign up for one by clicking **"Create an account"**. If you have forgotten your password, click on **"Forgot Password"** to create a new one. Once you've entered your Syncfusion email and password, click Next.



The screenshot shows the 'Syncfusion Essential Studio for JavaScript' offline installer window. The window has a dark theme and includes the Syncfusion logo in the top left. The title bar shows standard window controls. The main content area displays the product name, version (25.1.35), date (March 14, 2024), and supported frameworks (Angular, React, and Vue). A message thanks the user for choosing the installer and prompts them to enter required information. There are two tabs: 'Login To Install' (selected) and 'Use Unlock Key'. The 'Login To Install' tab contains a login form with fields for 'Email *' and 'Password *', a 'Forgot Password?' link, and a 'Don't have an account? Create an account' link. Below the login form is a checkbox for 'I agree to the License Terms and Privacy Policy'. A 'Next' button is located at the bottom right of the window.

Syncfusion

Essential Studio for JavaScript

Version: 25.1.35, Date: March 14, 2024

Supported frameworks: Angular, React, and Vue.

Thank you for choosing Syncfusion Essential Studio. Please enter the required information to proceed further.

Login To Install **Use Unlock Key**

Email *

Password *

[Forgot Password?](#)

Don't have an account? [Create an account](#)

☐ I agree to the [License Terms](#) and [Privacy Policy](#)

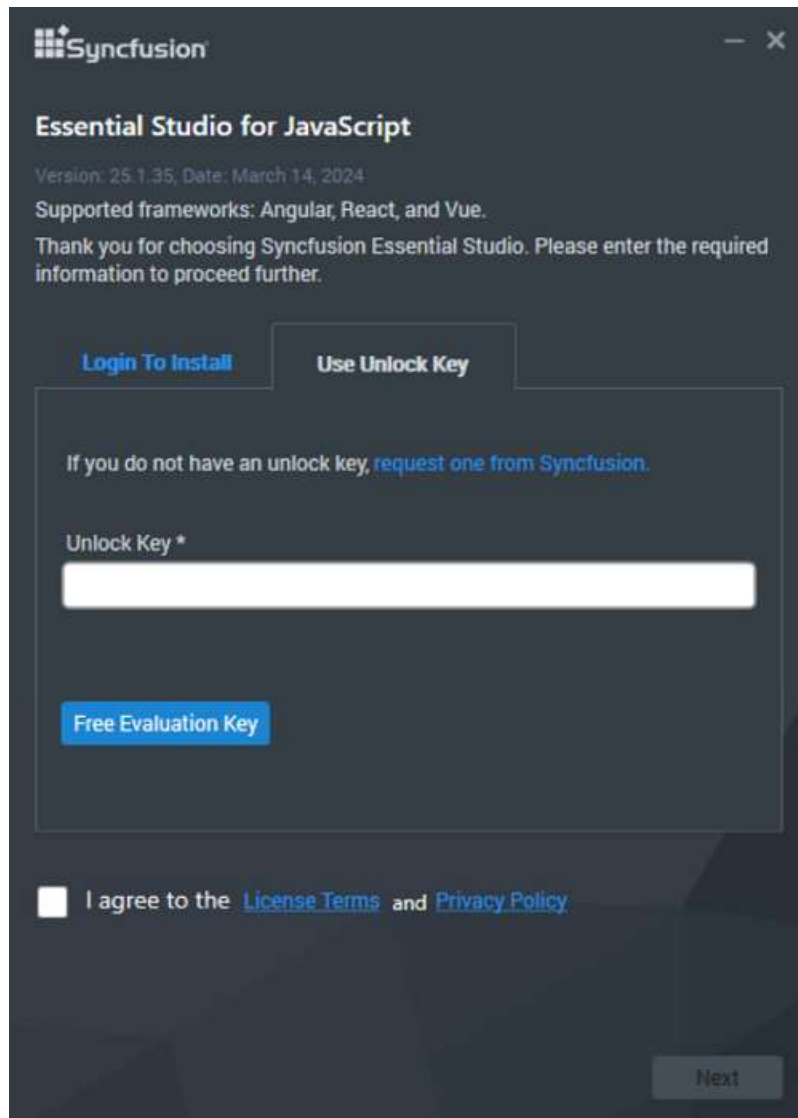
Next

Use Unlock Key

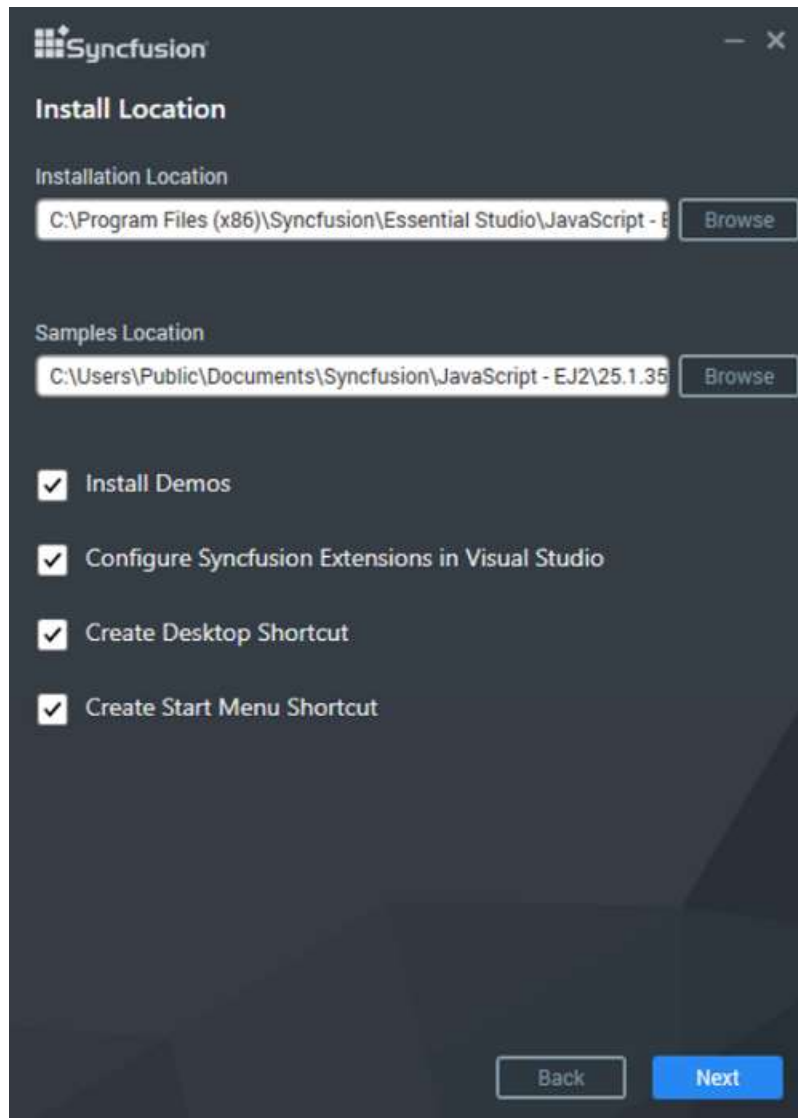
Unlock keys are used to unlock the Syncfusion offline installer, and they are platform and version specific. You should use either Syncfusion licensed or trial Unlock key to unlock Syncfusion JavaScript – EJ2 installer.

The trial unlock key is only valid for 30 days, and the installer will not accept an expired trial key.

To learn how to generate an unlock key for both trial and licensed products, see [this](#) Knowledge Base article.

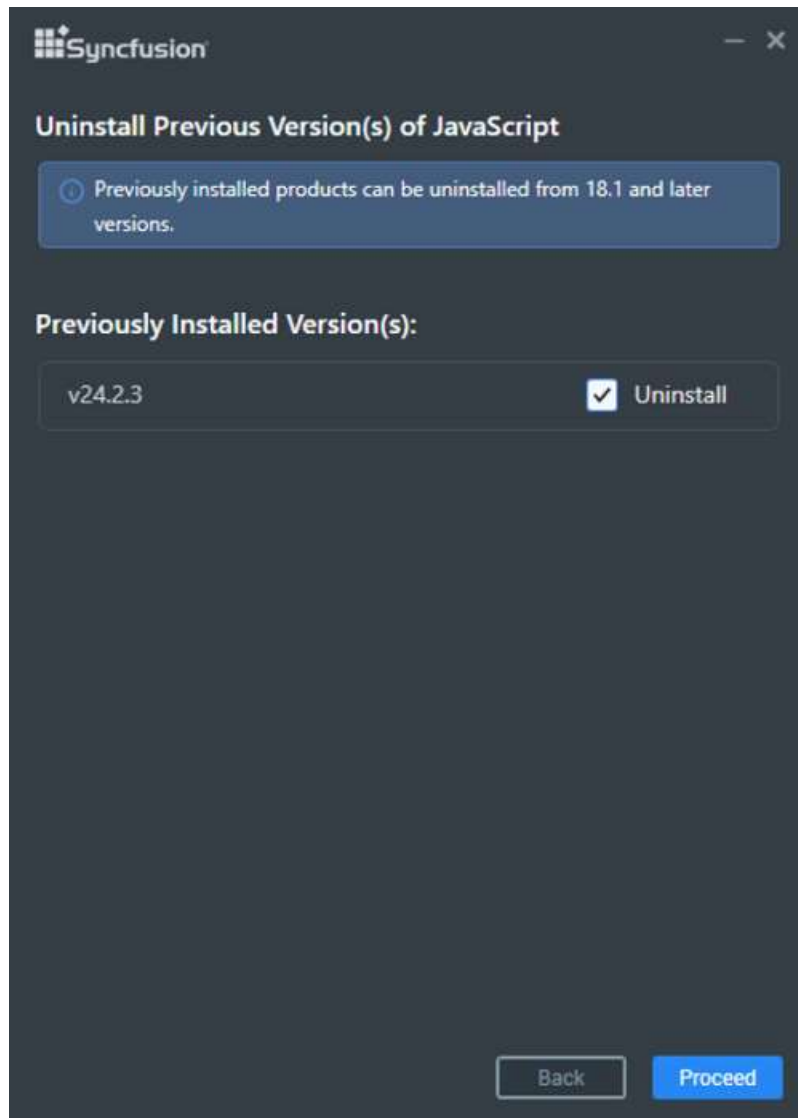


3. After reading the License Terms and Privacy Policy, check the **"I agree to the License Terms and Privacy Policy"** check box. Click the Next button.
4. Change the install and sample locations here. You can also change the Additional settings. Click Next\Install to install with the default settings.



Additional Settings

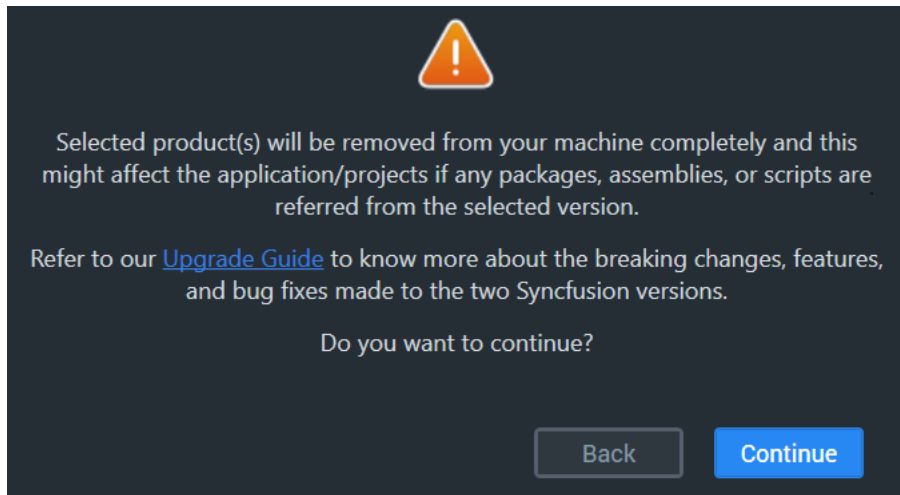
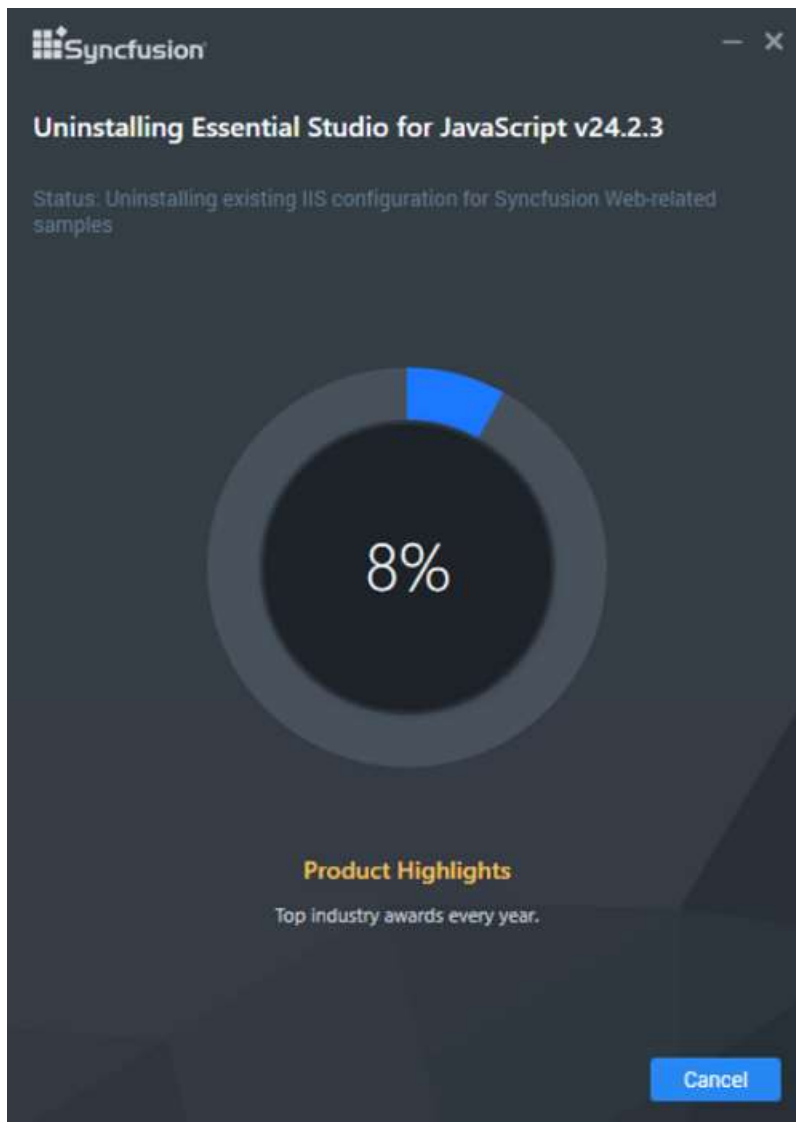
- Select the **Install Demos** check box to install Syncfusion samples, or leave the check box unchecked, if you do not want to install Syncfusion samples
 - Select the **Configure Syncfusion Extensions controls in Visual Studio** checkbox to configure the Syncfusion Extensions in Visual Studio or clear this check box when you do not want to configure the Syncfusion Extensions in Visual Studio.
 - Check the **Create Desktop Shortcut** checkbox to add a desktop shortcut for Syncfusion Control Panel
 - Check the **Create Start Menu Shortcut** checkbox to add a shortcut to the start menu for Syncfusion Control Panel
5. If any previous versions of the current product is installed, the **Uninstall Previous Version(s)** wizard will be opened. Select Uninstall checkbox to uninstall the previous versions and then click the Proceed button.

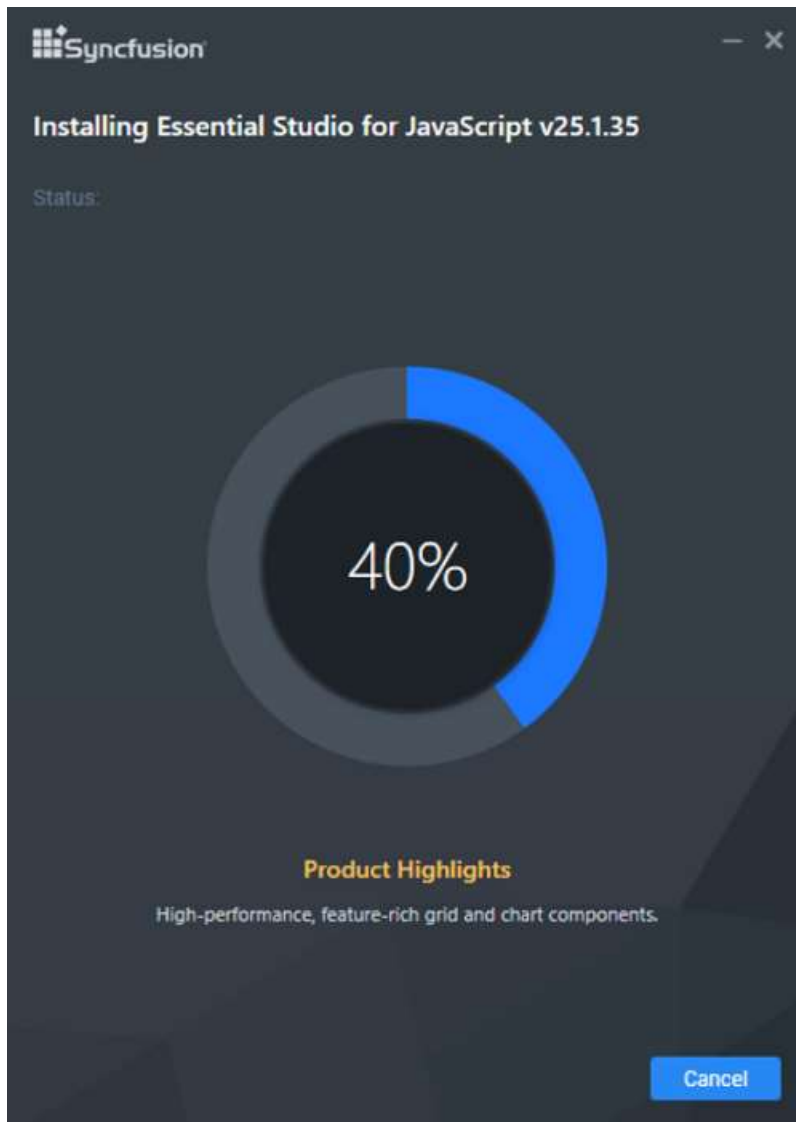


Note: From the 2021 Volume 1 release, Syncfusion has added the option to uninstall previous versions from 18.1 while installing the new version.

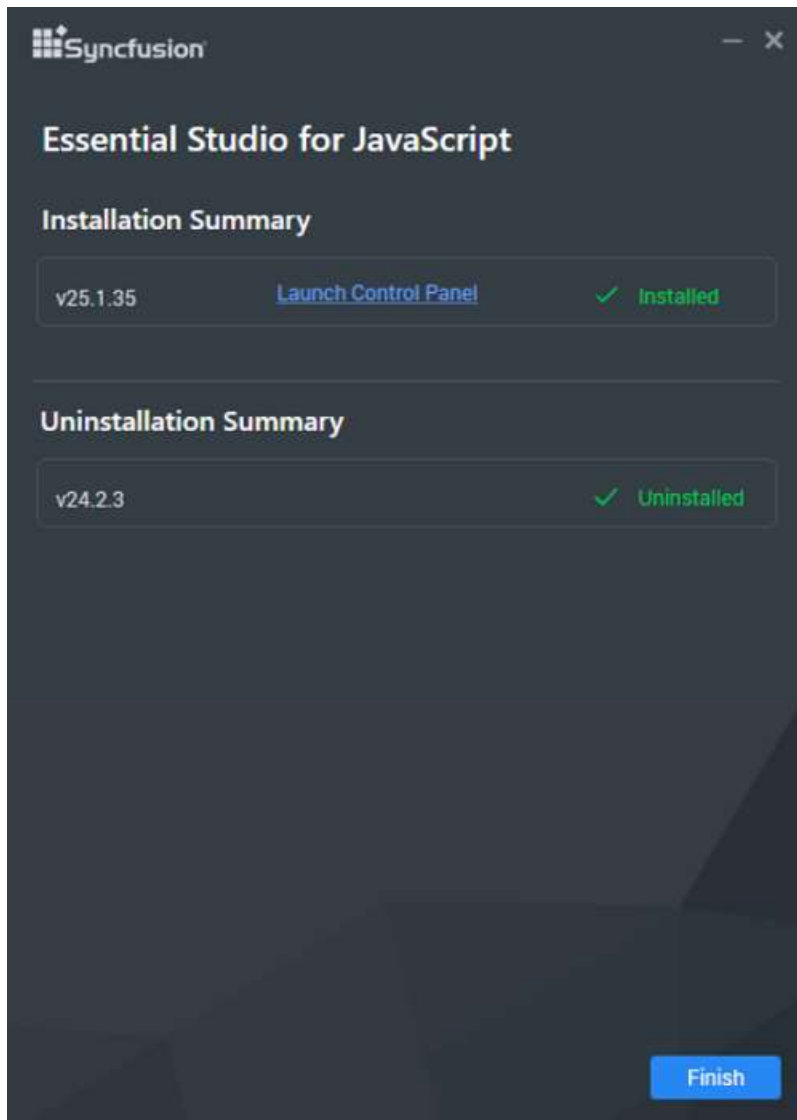
Note: If any version is selected to uninstall, a confirmation screen will appear; if continue is selected, the Progress screen will display the uninstall and install progress, respectively. If none of the versions are chosen to be uninstalled, only the installation progress will be displayed.

Confirmation Alert

**Uninstall Progress:**

Install Progress

Note: The Completed screen is displayed once the JavaScript – EJ2 product is installed. If any version is selected to uninstall, The completed screen will display both install and uninstall status.



6. After installing, click the Launch Control Panel link to open the Syncfusion Control Panel.
7. Click the Finish button. Your system has been installed with the Syncfusion Essential Studio JavaScript – EJ2 product.

Installing in silent mode

The Syncfusion Essential Studio JavaScript – EJ2 Installer supports installation and uninstallation via the command line.

Command line installation

To install through the Command Line in Silent mode, follow the steps below.

1. Run the Syncfusion JavaScript – EJ2 installer by double-clicking it. The Installer Wizard automatically opens and extracts the package.
2. The file `syncfusionejs2_(version).exe` file will be extracted into the Temp directory.
3. Run `%temp%`. The Temp folder will be opened. The `syncfusionejs2_(version).exe` file will be located in one of the folders.

4. Copy the extracted syncfusionejs2_(version).exe file in local drive.
5. Exit the Wizard.
6. Run Command Prompt in administrator mode and enter the following arguments.

Arguments: "installer file path\SyncfusionEssentialStudio(product)_(version).exe" /Install silent /UNLOCKKEY:"(product unlock key)" [/log "{Log file path}"] [/InstallPath:{Location to install}] [/InstallSamples:{true/false}] [/InstallAssemblies:{true/false}] [/UninstallExistAssemblies:{true/false}] [/InstallToolbox:{true/false}]

Note: [...] – Arguments inside the square brackets are optional.

Example: "D:\Temp\syncfusionejs2x.x.x.x.exe" /Install silent /UNLOCKKEY:"product unlock key" /log "C:\Temp\EssentialStudioPlatform.log" /InstallPath:C:\Syncfusion\x.x.x.x /InstallSamples:true /InstallAssemblies:true /UninstallExistAssemblies:true /InstallToolbox:true

7. Essential Studio for JavaScript (Essential JS2) is installed.

Note: x.x.x.x should be replaced with the Essential Studio version and the Product Unlock Key needs to be replaced with the Unlock Key for that version.

Command line uninstallation

Syncfusion Essential JavaScript – EJ2 can be uninstalled silently using the Command Line.

1. Run the Syncfusion JavaScript – EJ2 installer by double-clicking it. The Installer Wizard automatically opens and extracts the package.
2. The syncfusionejs2_(version).exe file will be extracted into the Temp directory.
3. Run %temp%. The Temp folder will be opened. The syncfusionejs2_(version).exe file will be located in one of the folders.
4. Copy the extracted syncfusionejs2_(version).exe file in local drive.
5. Exit the Wizard.
6. Run Command Prompt in administrator mode and enter the following arguments.

Arguments: "Copied installer file path\ syncfusionejs2_(version).exe" /uninstall silent

Example: "D:\Temp\syncfusionejs2_x.x.x.x.exe" /uninstall silent

7. Essential Studio for JavaScript (Essential JS2) is uninstalled.

Mac Installer

Download JavaScript – EJ2 Mac Installer

The Syncfusion JavaScript - EJ2 Mac installer can be downloaded from the Syncfusion website. You can either download the licensed installer or try our trial installer depending on your license.

- Trial Installer
- Licensed Installer

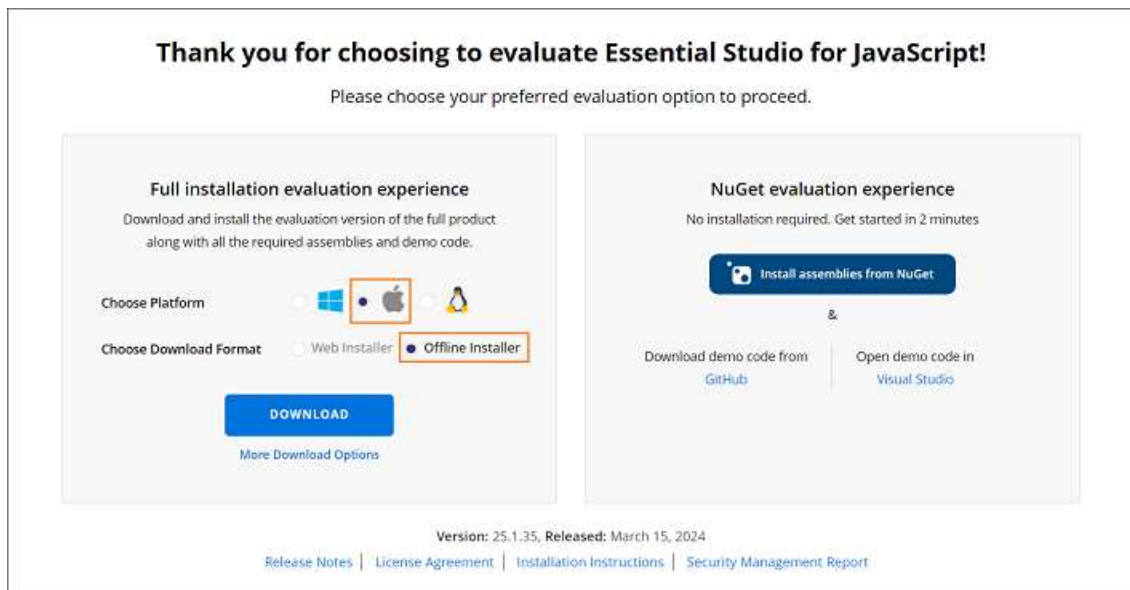
Download the trial version

Our 30-day trial can be downloaded in two ways.

- Download Free Trial Setup
- Start Trials if using components through [npm](#)

Download free trial setup

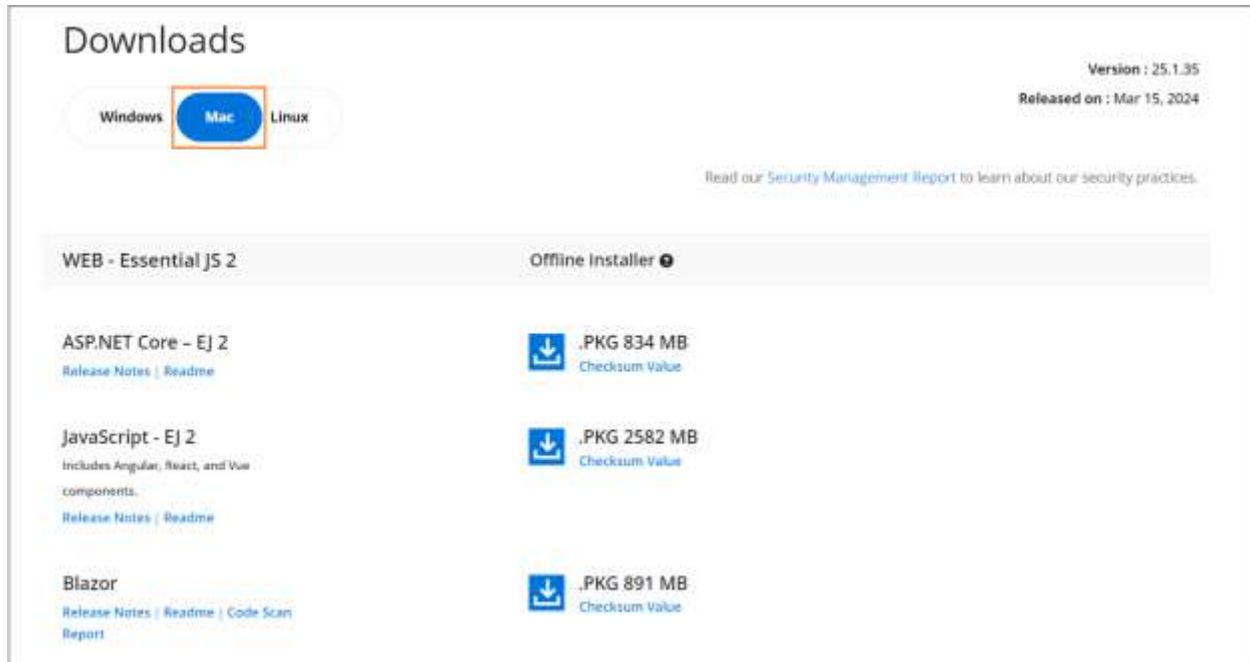
1. You can evaluate our 30-day free trial by visiting the [Download Free Trial](#) page and select the JavaScript platform.
2. After completing the required form or logging in with your registered Syncfusion account, you can download the JavaScript - EJ2 trial installer from the confirmation page. (See the screenshot below.)



3. With a trial license, only the latest version's trial installer can be downloaded.
4. After downloading, the Syncfusion JavaScript - EJ2 trial installer can be unlocked using either the trial unlock key or the Syncfusion registered login credential. More information on generating an unlock key can be found in [this](#) article.
5. Before the trial expires, you can download the trial installer at any time from your registered account's [Trials & Downloads](#) page (as shown in below screenshot.)



6. Click the More Download Options (element 2 in the above screenshot) button to get the Essential Studio JavaScript - EJ2 Offline trial installer which is available in PKG format.



Start trials if using components through [npm](#)

You should initiate an evaluation if you have already obtained our components through [npm](https://www.npmjs.com/~syncfusionorg)

1. You can start your 30-day free trial for JavaScript - EJ2 from the [Start Trial](#) page from your account.



2. To access this page, you must sign up\log in with your Syncfusion account.
3. Begin your trial by selecting the WinUI product.

Note: If you've already used the trial products and they haven't expired, you won't be able to start the trial for the same product again.

4. After you've started the trial, go to the [Trials & Downloads](#) page to get the latest version trial installer. You can generate the [unlock key](#) and [license key](#) here at any time before the trial period expires. (as shown in below screenshot.)

Trial Downloads and Unlock Keys

Starting with v20.1.0.47 (2022 Vol. 1), we're providing licensing support for Essential Studio for JavaScript (Essential JS 2). Please refer to this [help topic](#) for more information.

Windows Forms [REDACTED]

Trial Version : [REDACTED]

[What's New](#) | [Release Notes](#) | [Get License Key](#) ⓘ | [Get Unlock Key](#) ⓘ | [Security Management Report](#)

[Download](#)

1

[More Download Options](#)

2

5. You can find your current active trial products on the [Trials & Downloads](#) page.

Download the license version

1. Syncfusion licensed products will be available in the [License & Downloads](#) page under your registered Syncfusion account.
2. You can view all the licenses (both active and expired) associated with your account.
3. You can download JavaScript - EJ2 Mac licensed installer by going to More Downloads Options (element 3 in the screenshot below).

License Downloads and Unlock Keys

Essential Studio **License SKU Name** [REDACTED]

[Download Older Versions](#) 2

Latest Official Release : [REDACTED]

[What's New](#) | [Generate License File](#) ⓘ | [Get Unlock Key](#) ⓘ

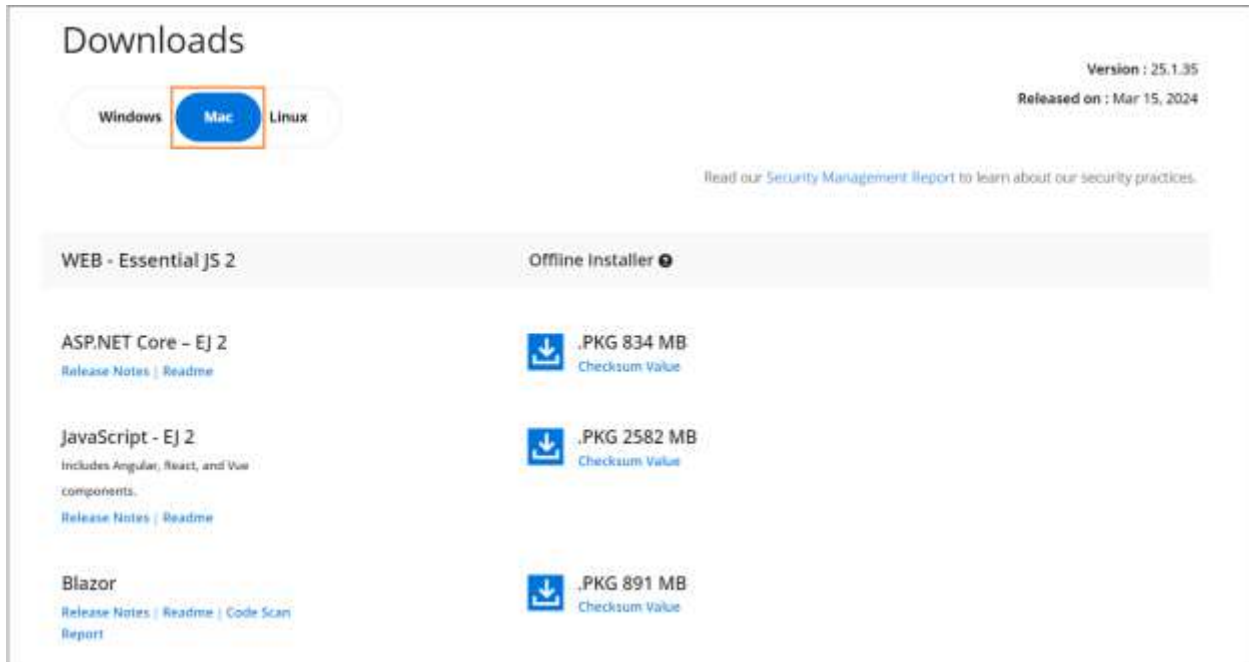
[Download](#)

1

[More Download Options](#)

3

4. Unlock key is not required to install the Syncfusion JavaScript - EJ2 Mac trial installer.
5. For Mac OS, PKG formats is available for download.

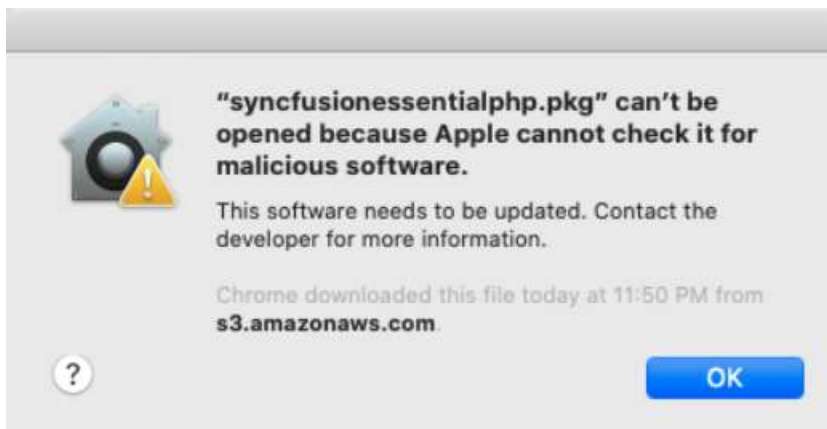


You can also refer to the [JavaScript - EJ2 Mac installer](#) links for step-by-step installation guidelines.

Installing Syncfusion JavaScript – EJ2 Mac Installer

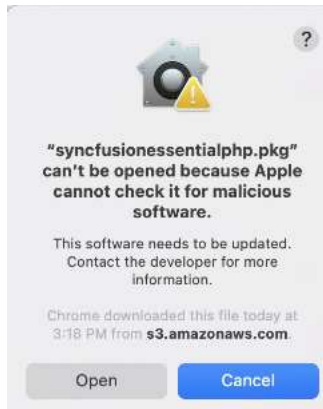
Steps to resolve the warning message in Catalina OS or later

While running Essential Studio JavaScript - EJ2 Mac Installer on Catalina MacOS or later, the below alert will be displayed.



If you receive this alert, follow the below steps for the easiest solution.

1. Right-click the downloaded pkg file.
2. Select the "Open With" option and choose "Installer (Default)". The following pop-up appears.

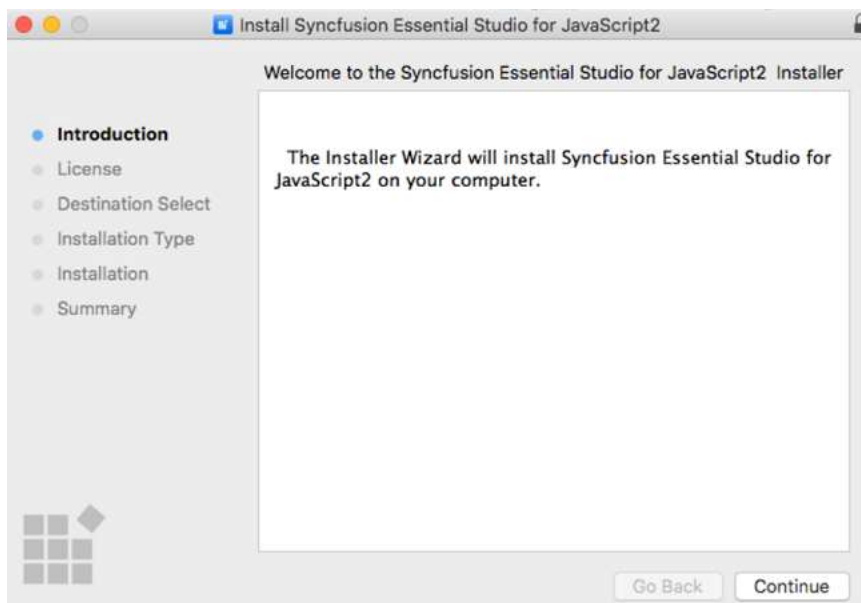


3. When you click "Open" the installer window will be opened.

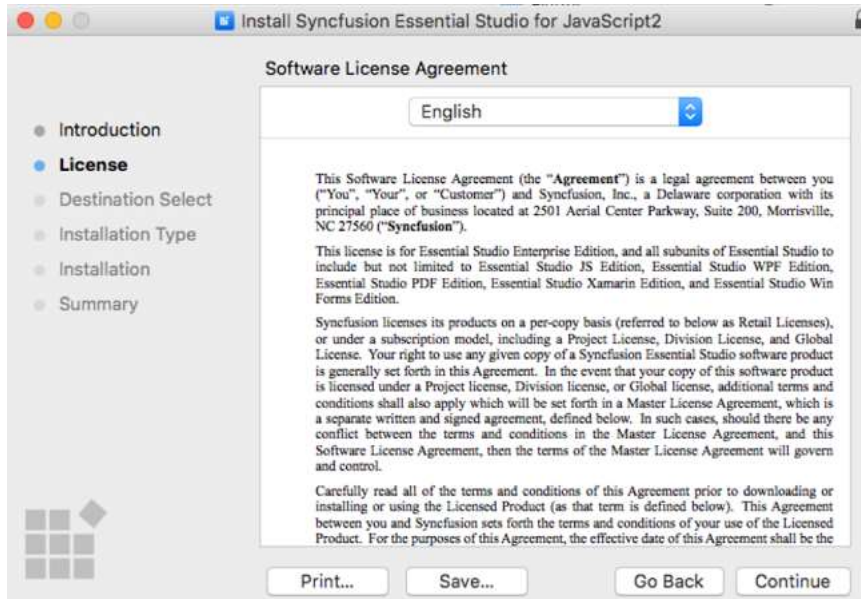
Step-by-Step installation

The steps below show how to install the Essential Studio JavaScript - EJ2 Mac installer.

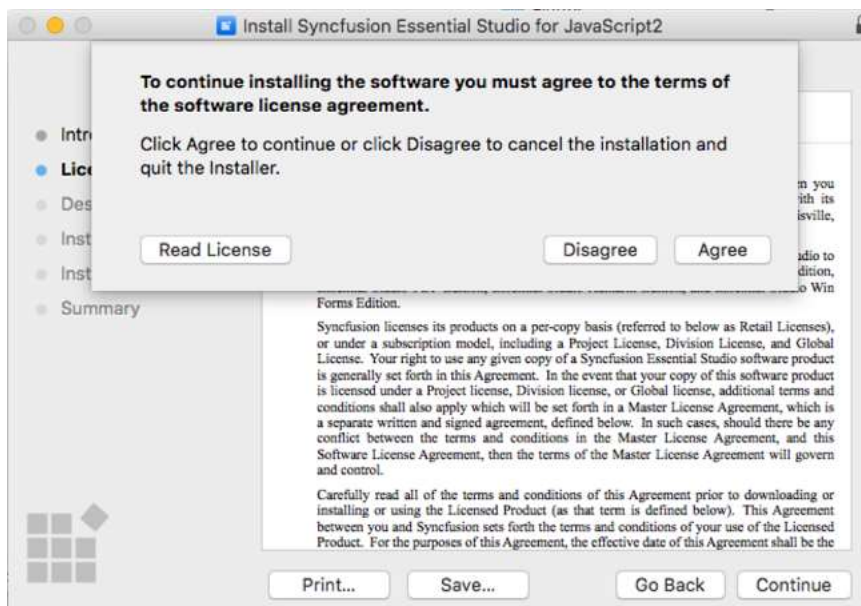
1. Double-click the Syncfusion Essential Studio JavaScript - EJ2 Mac installer(.pkg) file. The installer Wizard opens. Click Continue.



2. The Software License Agreement wizard will appear. Click the Continue button.

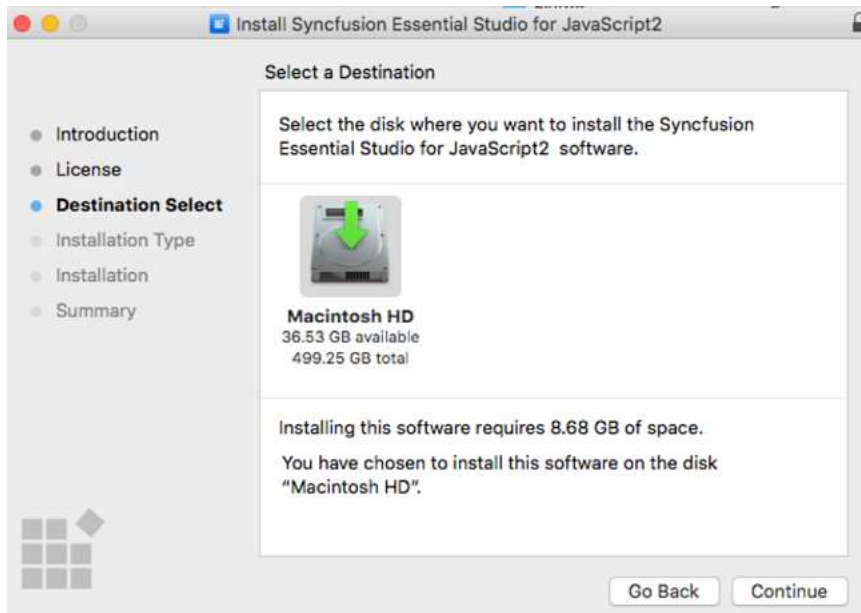


3. The License Agreement's Confirmation window will appear. If you have read the Software License Agreement, click **Agree**.

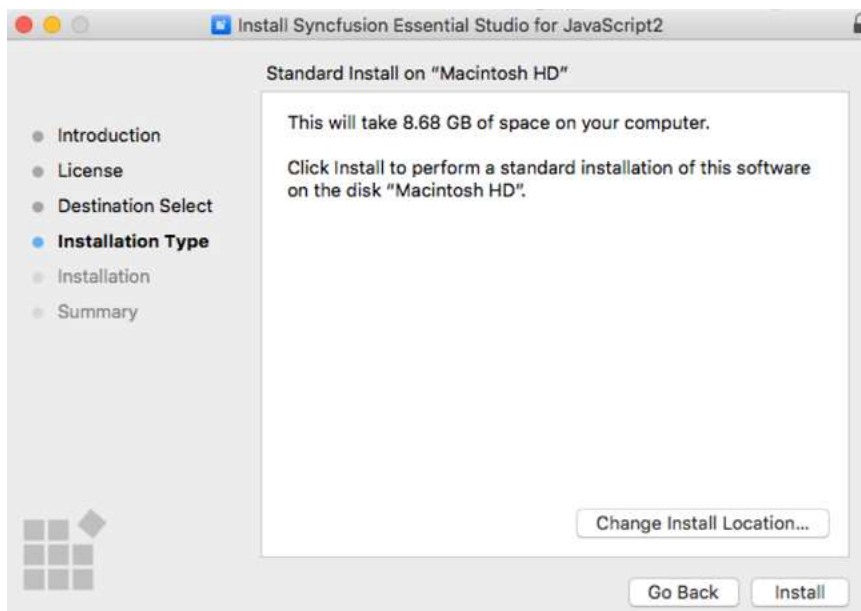


Note: The Unlock key is not required to install the Mac installer. The Syncfusion Essential Studio JavaScript - EJ2 Mac installer can be used for development purposes without registering the Unlock key.

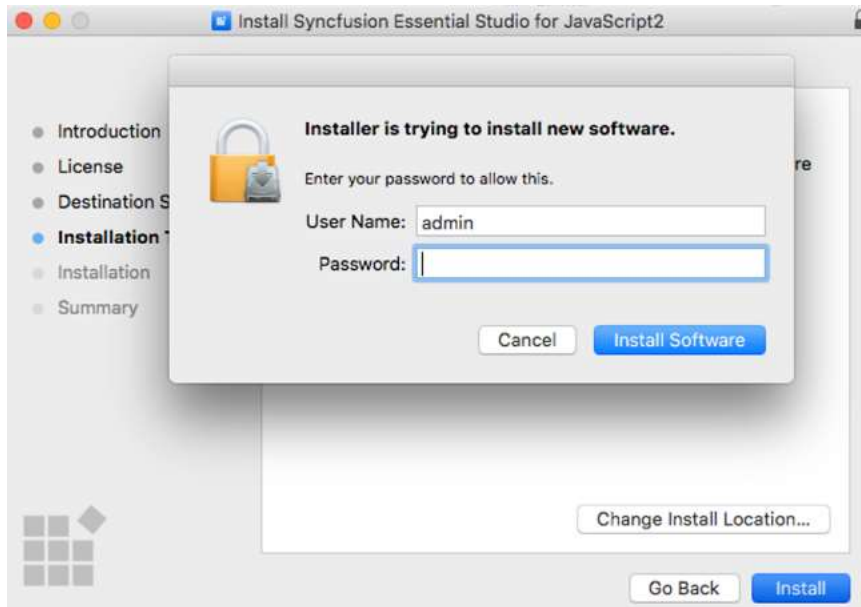
4. The Destination select wizard will appear. You can choose which disc to install the Syncfusion Essential Studio for JavaScript - EJ2 Mac installer on here.



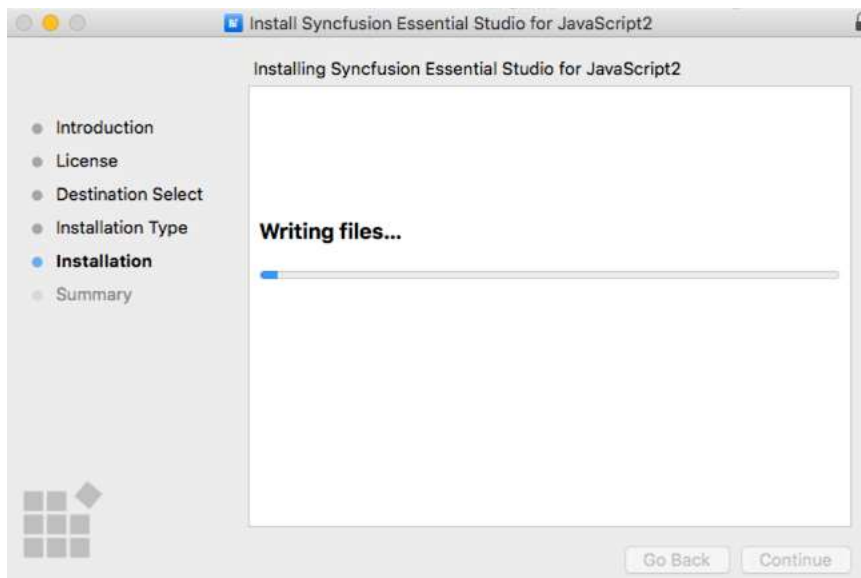
5. The Installation Type wizard will appear. Click Install to begin the standard installation of the Syncfusion Essential Studio JavaScript - EJ2 Mac installer.



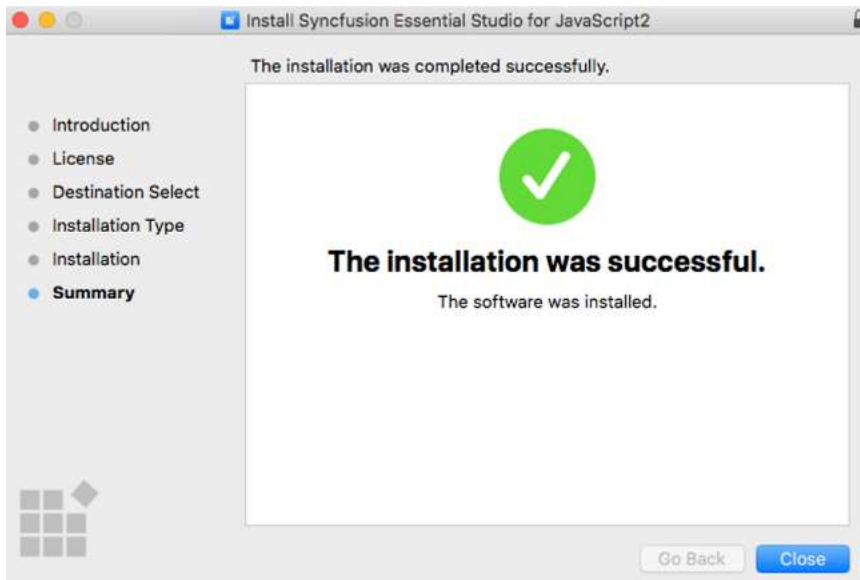
6. The Authentication window will appear. To begin the installation, enter the Mac machine's password and click **Install Software**.



7. The installation process will begin on your machine.

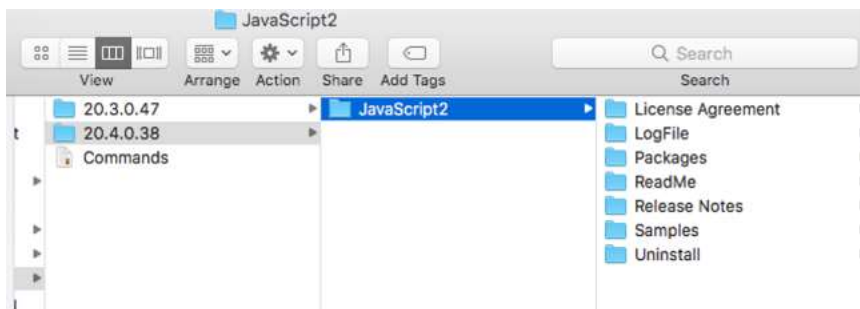


8. Once the installation is complete, the completed screen will be displayed. To exit the installation wizard, click Close.



By default, Mac installer will install the files in following location.

Location: {Documents}\Syncfusion\ {version}\ {platform}



License key registration in samples

After the installation, the license key is required to register the demo source that is included in the Mac installer. To learn about the steps for license registration for the JavaScript - EJ2 Mac installer, please refer to this.

- [Register Syncfusion License key in the project](#)
- [Register the license key using the npx command](#)

Linux Installer

Download Syncfusion JavaScript Linux Installer

The Syncfusion installer can be downloaded from the [Syncfusion](#) website. You can either download the licensed installer or try our trial installer depending on your license.

- Trial Installer - Licensed Installer

You can download the Syncfusion installer from [Syncfusion.com](#) website

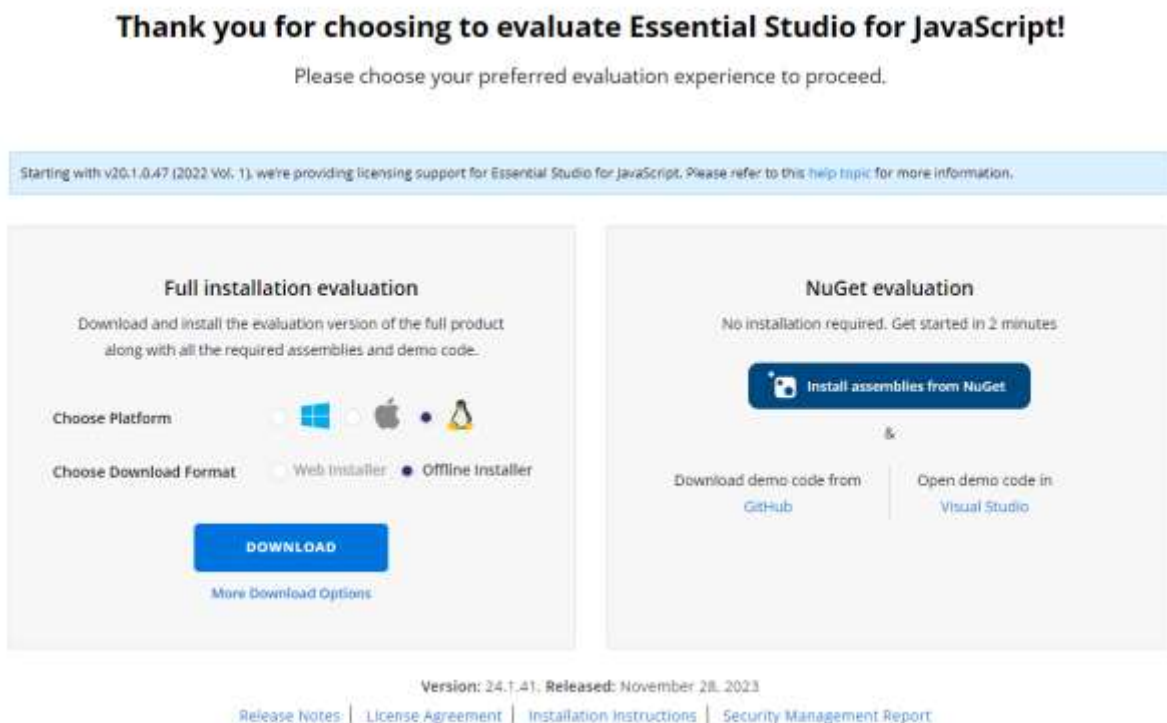
Download the Trial Version

Our 30-day trial can be downloaded in two ways.

- Download Free Trial Setup
- Start Trials if using components through [NuGet.org](https://www.nuget.org)

Download Free Trial Setup

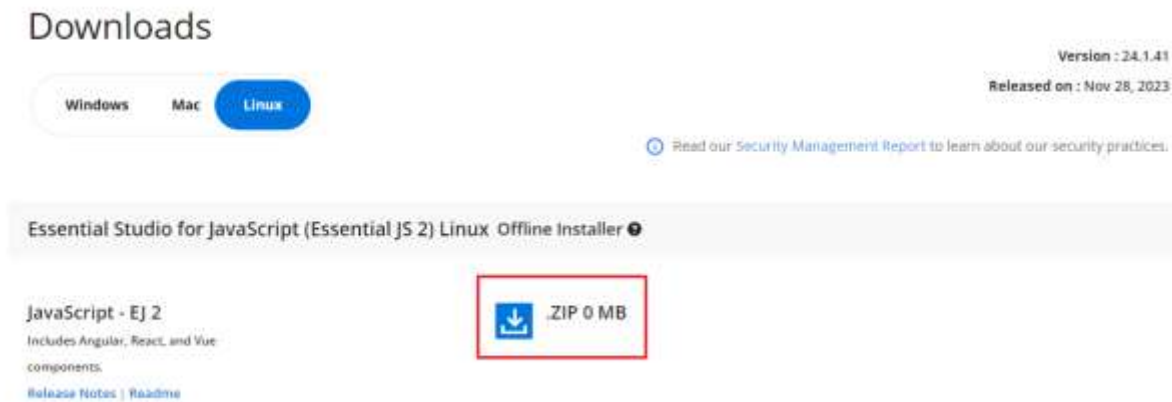
1. You can evaluate our 30-day free trial by visiting the [Download Free Trial](#) page and select the product
2. After completing the required form or logging in with your registered Syncfusion account, you can download the trial installer from the confirmation page. (as shown in below screenshot.)



3. With a trial license, only the latest version's trial installer can be downloaded.
4. Unlock key is not required to install the Syncfusion JavaScript Linux trial installer.
5. Before the trial expires, you can download the trial installer at any time from your registered account's [Trials & Downloads](#) page (as shown in below screenshot.)



6. Click the More Download Options (element 2 in the above screenshot) button to get the JavaScript Product Offline trial installer which is available in ZIP format.

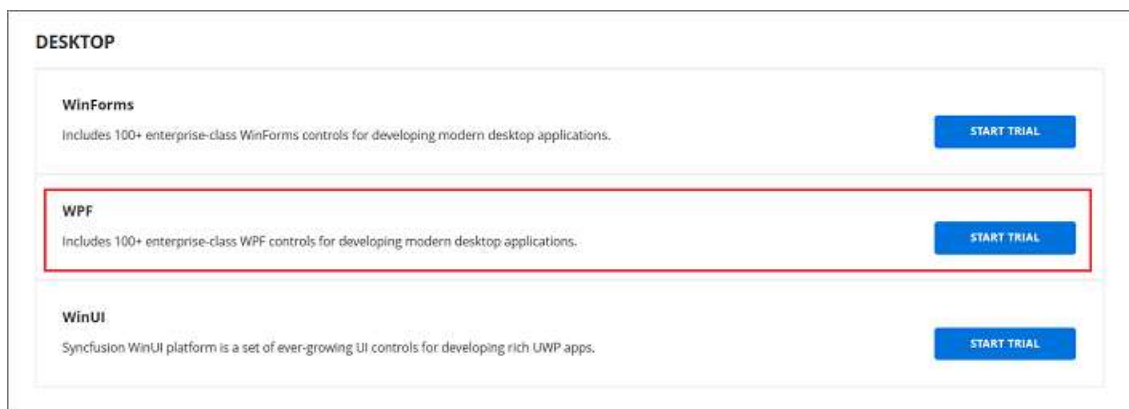


Start Trials if using components through [NuGet.org](https://www.nuget.org/packages?q=syncfusion)

You should initiate an evaluation if you have already obtained our components through [NuGet.org](https://www.nuget.org/packages?q=syncfusion)

1. You can start your 30-day free trial from the [Start Trial](#) page from your account.

Note: You can generate the license key for your active trial products from [Trials & Downloads](#) page. This license key will be mandatory to use our trial products in your application. To know more about License key, refer this [help topic](#).



2. To access this page, you must sign up\log in with your Syncfusion account.
3. Begin your trial by selecting the Syncfusion product.

Note: If you've already used the trial products and they haven't expired, you won't be able to start the trial for the same product again.

4. After you've started the trial, go to the [Trials & Downloads](#) page to get the latest version trial installer. You can generate the [unlock key](#) and [license key](#) here at any time before the trial period expires. (as shown in below screenshot.)



5. You can find your current active trial products on the [Trials & Downloads](#) page.

Download the License Version

1. Syncfusion licensed products will be available in the [License & Downloads](#) page under your registered Syncfusion account.
2. You can view all the licenses (both active and expired) associated with your account.
3. You can download JavaScript Linux licensed installer by going to More Downloads Options (element 3 in the screenshot below).



4. Unlock key is not required to install the Syncfusion JavaScript Linux trial installer.
5. For Linux OS, ZIP formats is available for download.

Downloads

Windows

Mac

Linux

Version : 24.1.41
Released on : Dec 18, 2023

[Read our Security Management Report](#) to learn about our security practices.

WEB - Essential JS 2

Offline Installer

Blazor

[Release Notes](#) | [Readme](#)

.ZIP 652 MB
Checksum Value

ASP.NET Core - EJ 2

[Release Notes](#) | [Readme](#)

.ZIP 1092 MB
Checksum Value

JavaScript - EJ 2

[Release Notes](#) | [Readme](#)

.ZIP 2066 MB
Checksum Value

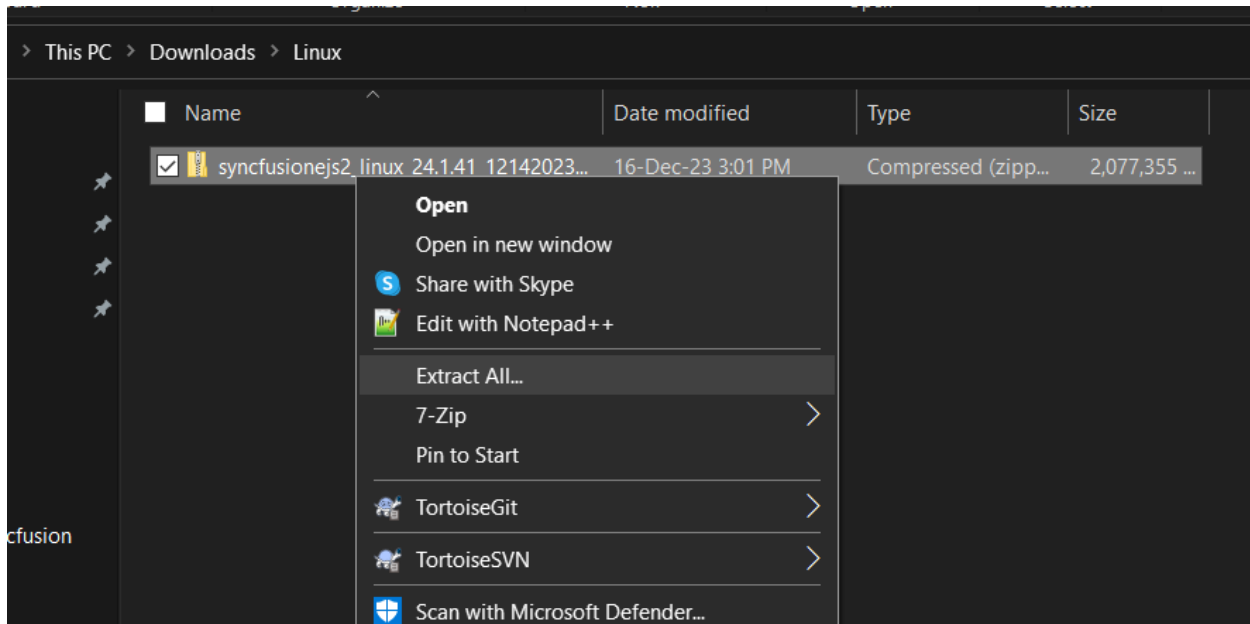
You can also refer to the [JavaScript Linux installer](#) links for step-by-step installation guidelines.

Installing Syncfusion JavaScript Linux installer

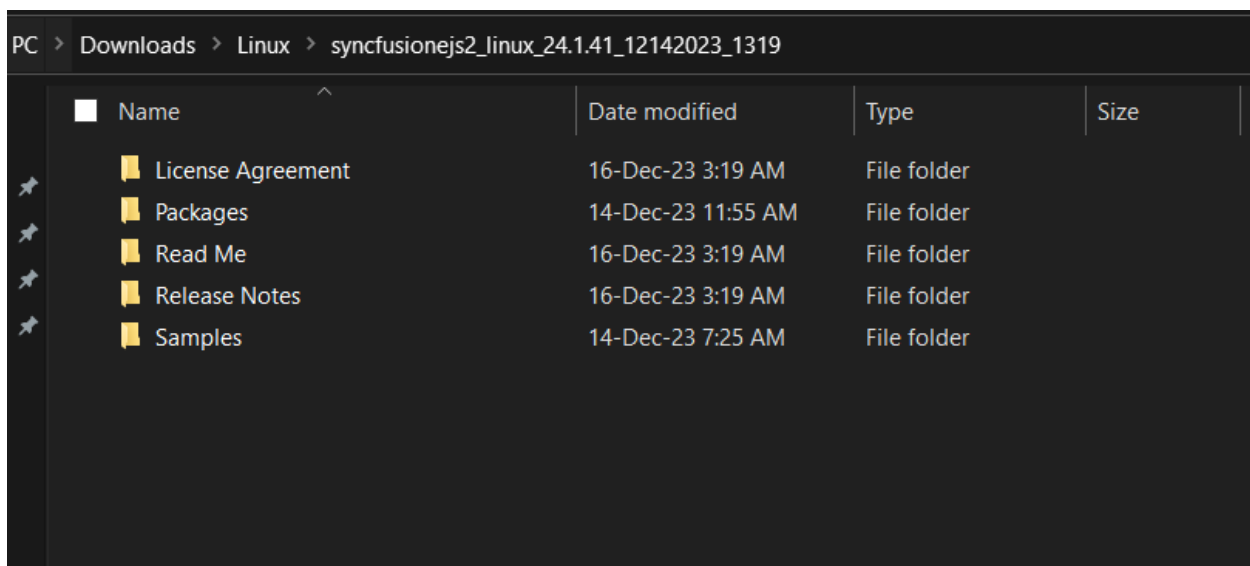
Step-by-Step Installation

The steps below show how to install JavaScript Linux installer.

1. Extract the Syncfusion JavaScript Linux installer(.zip) file. The files are extracted in your machine.



2. The Linux zip file contains the following folders.



Note: The Unlock key is not required to install the Linux installer.

4. You can launch the demo source and use the NuGet packages included in the Linux installer.

License key registration in samples

After the installation, the license key is required to register the demo source that is included in the Linux installer. To learn about the steps for license registration for the JavaScript - EJ2 linux installer, please refer to this.

- [Register Syncfusion License key in the project](#)
- [Register the license key using the npx command](#)

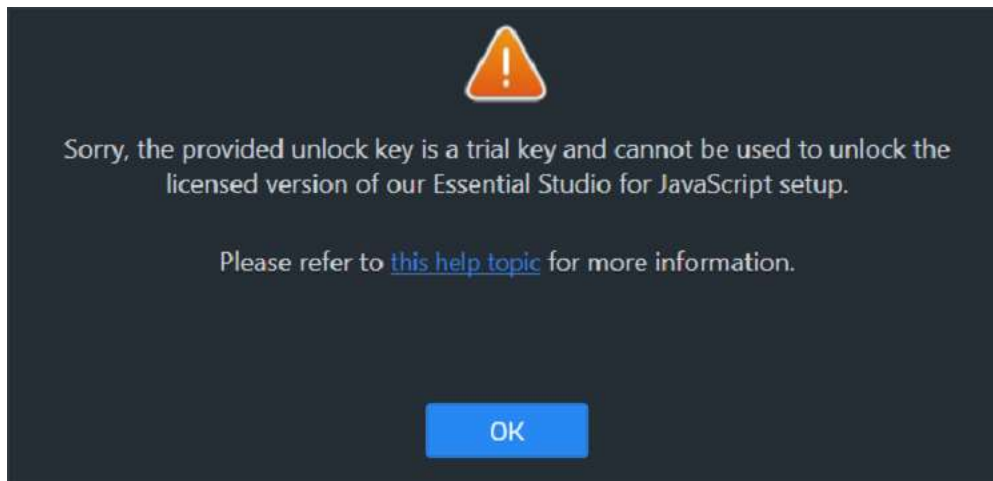
Common Installation Errors

This article describes the most common installation errors, as well as the causes and solutions to those errors.

- Unlocking the license installer using the trial key
- License has expired
- Unable to find a valid license or trial
- Unable to install because of another installation
- Unable to install due to controlled folder access

Unlocking the license installer using the trial key

Error Message: Sorry, the provided unlock key is a trial unlock key and cannot be used to unlock the licensed version of our Essential Studio for JavaScript installer.



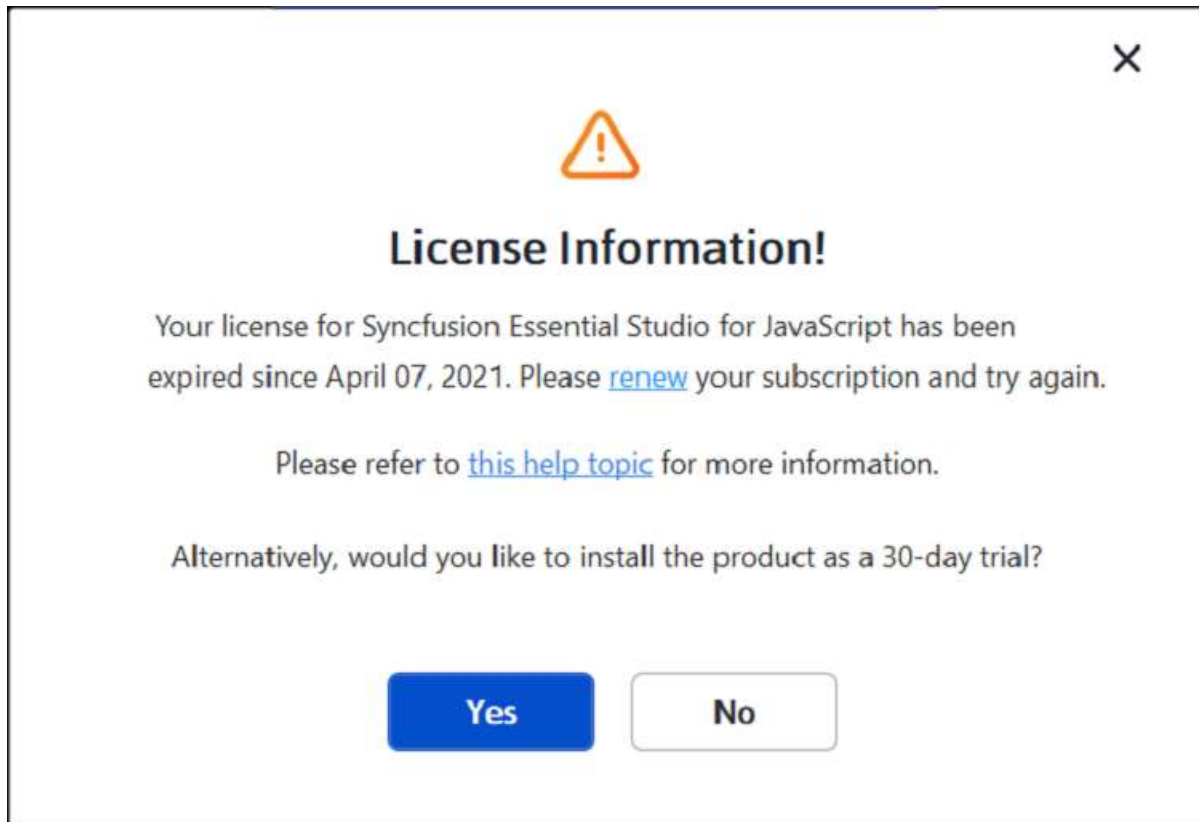
Reason
 You are attempting to use a Trial unlock key to unlock the licensed installer.

Suggested solution
 Only a licensed unlock key can unlock a licensed installer. So, to unlock the Licensed installer, use the Licensed unlock key. To generate the licensed unlock key, refer to [this](#) article.

License has expired

Error Message: Your license for Syncfusion Essential Studio for JavaScript – EJ2 has been expired since {date}. Please renew your subscription and try again.

Online Installer



Reason
 This error message will appear if your license has expired.

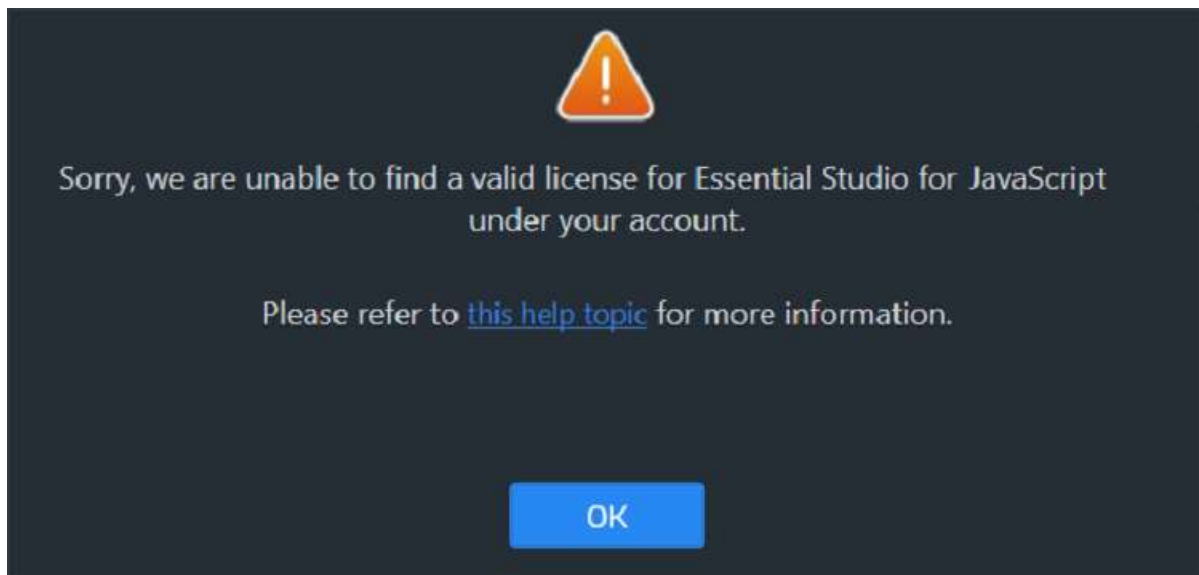
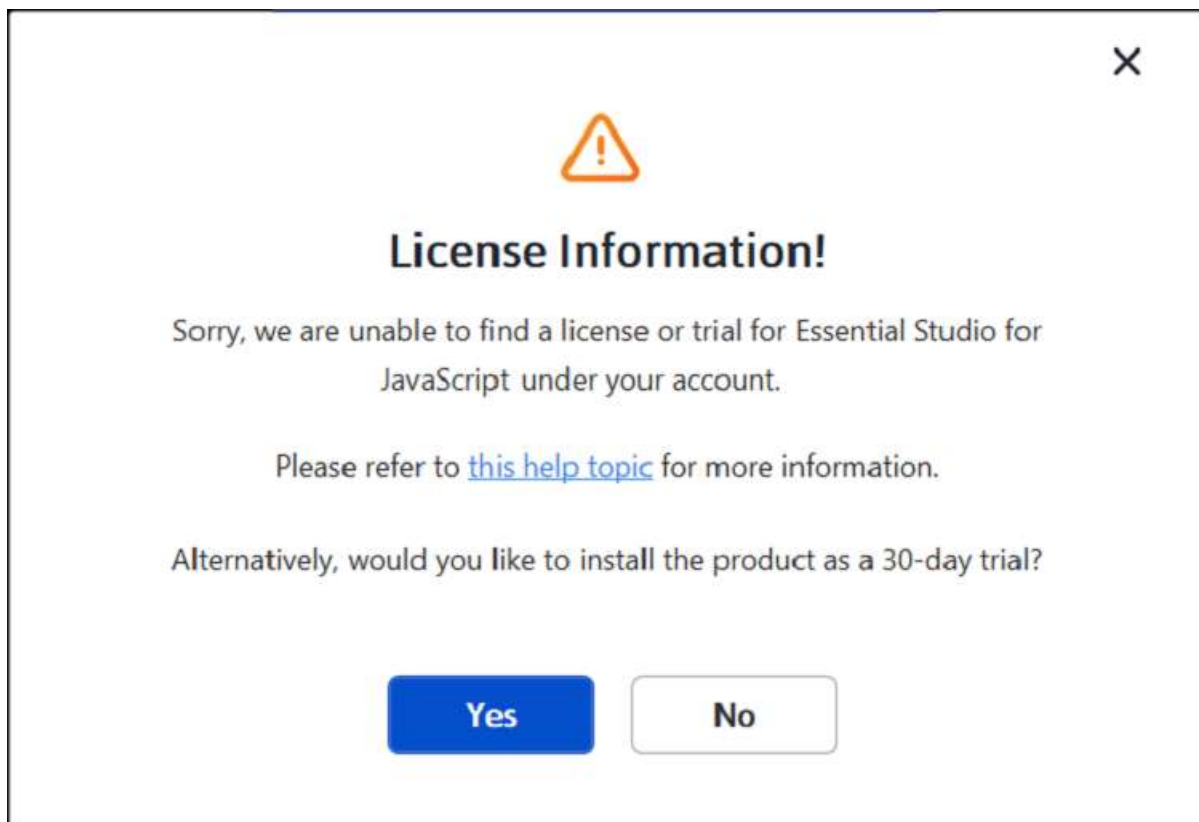
Suggested solution
 You can choose from the options below.

1. You can renew your subscription [here](#).
2. You can get a new license [here](#).
3. You can reach out to our sales team by emailing sales@syncfusion.com.
4. You can also extend the 30-day trial period after your trial license has expired.

[Unable to find a valid license or trial](#)

Error Message: Sorry, we are unable to find a valid license or trial for Essential Studio for JavaScript – EJ2 under your account.

Offline installer

**Online installer**

Reason
 The following are possible causes of this error:

The following are possible causes of this error:

- When your trial period expired
- When you don't have a license or an active trial
- You are not the license holder of your license

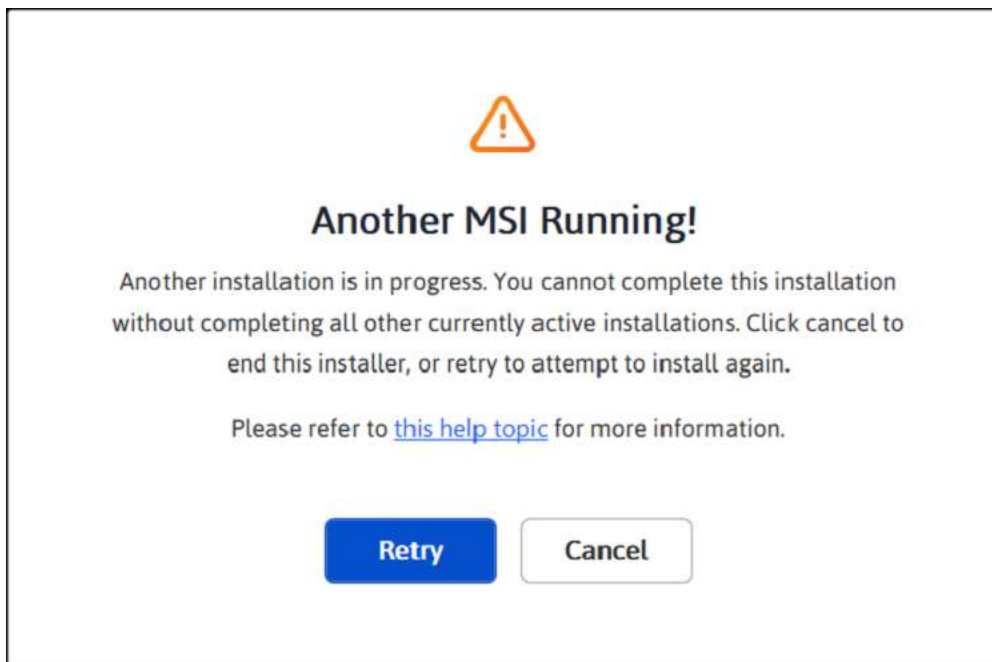
- Your account administrator has not yet assigned you a license.

Suggested solution

1. You can get a new license [here](#).
2. Contact your account administrator.
3. Send an email to clientrelations@syncfusion.com to request a license.
4. You can reach out to our sales team by emailing sales@syncfusion.com.

Unable to install because of another installation

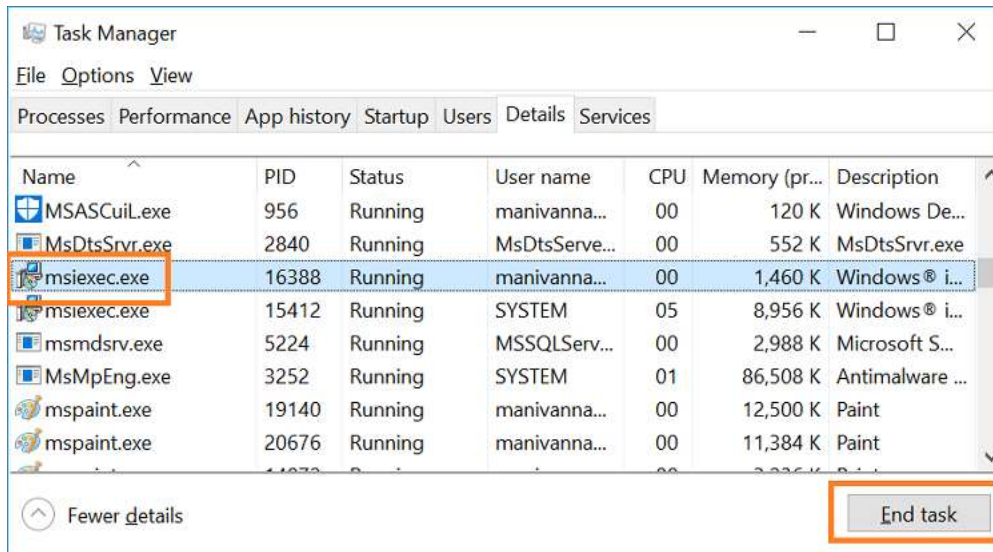
Error Message: Another installation is in progress. You cannot start this installation without completing all other currently active installations. Click cancel to end this installer or retry to attempt after currently active installation completed to install again.



Reason
 You are trying to install when another installation is already running in your machine.

Suggested solution
 Open and kill the msixexec process in the task manager and then continue to install Syncfusion. If the problem is still present, restart the computer and try Syncfusion installer.

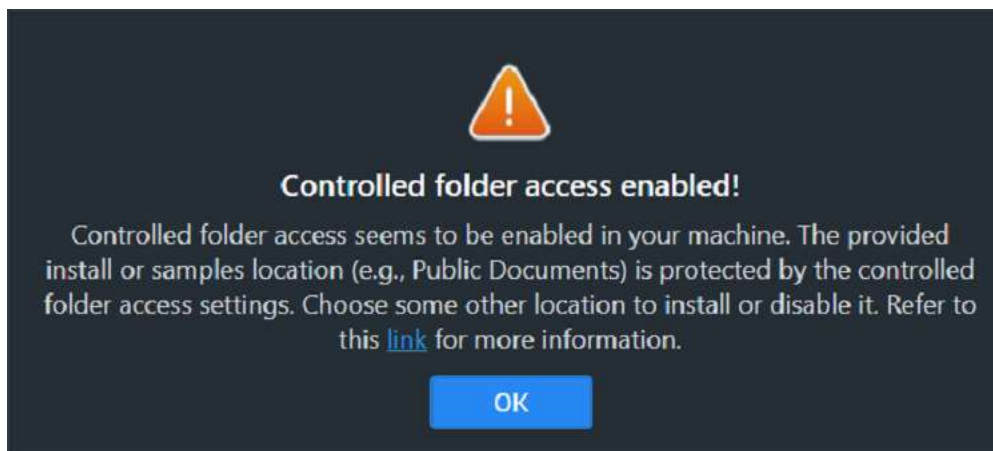
1. Open the Windows Task Manager.
2. Browse the Details tab.
3. Select the msixexec.exe and click **End task**.



Unable to install due to controlled folder access

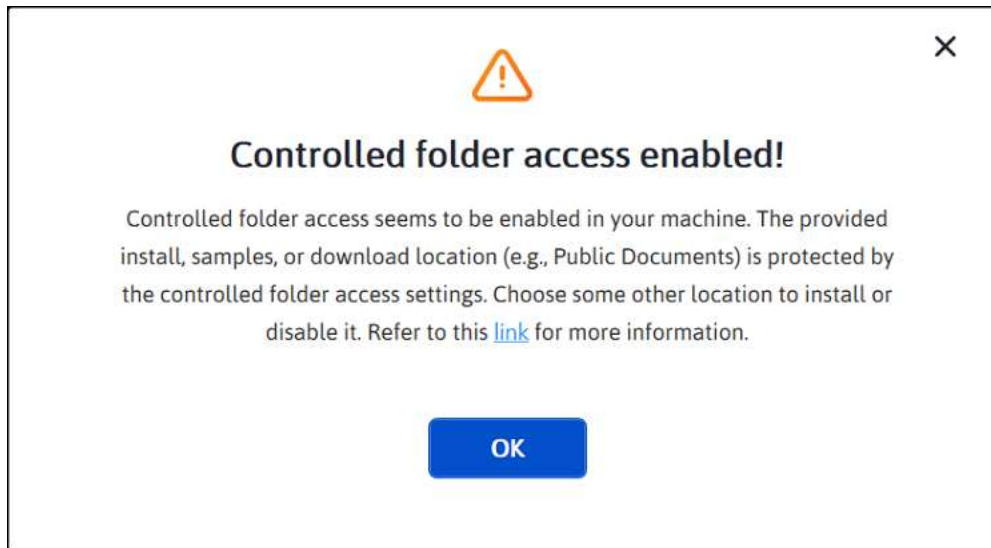
Offline

Error Message: Controlled folder access seems to be enabled in your machine. The provided install or samples location (e.g., Public Documents) is protected by the controlled folder access settings.



Online

Error Message: Controlled folder access seems to be enabled in your machine. The provided install, samples, or download location (e.g., Public Documents) is protected by the controlled folder access settings.



Reason
 You have enabled controlled folder access settings on your computer.

Suggested solution

Suggestion 1:

1. We will ship our demos in the public documents folder by default.
2. You have controlled folder access enabled on your machine, so our demos cannot be installed in the documents folder. If you need to install our demos in the Documents folder, follow the steps in this [link](#) and disable the controlled folder access.
3. You can enable this option after the installing our Syncfusion setup.

Suggestion 2:

1. If you do not want to disable controlled folder access, you can install our demos in another directory.

Upgrade

Release History

Every new release of Syncfusion includes exciting new features. Refer to the Syncfusion React [release notes](#) to know more about the changes in each releases.

See our [Upgrade Guide](#) for React to learn more about the following.

- Breaking changes
- Bug fixes
- Features
- Knows Issues between your current version and the latest version you are trying to upgrade.

Syncfusion React supported versions

React version compatibility

The following table represents the supported React versions by different Syncfusion React UI components releases.

Version	Syncfusion React components version
-----	-----
React v18.0	20.2.36 and above
React v17.0	18.3.50 and above
React v16.0	16.2.45 and above

Syncfusion version information

Syncfusion follows a quarterly release schedule, introducing new volumes every three months. To track these releases and their associated changes, Syncfusion React components utilize a sequence-based identifier system, employing the format **Major.Minor.Revision**. This system enables developers to easily monitor modifications made in each release.

For example, if the release package version is **22.1.34**, the version number can be interpreted as follows:

- **22** represents the **major release** version. This number changes every three months and encompasses significant updates, new features, as well as bug fixes and breaking changes.
- **1** corresponds to the **minor release** version. This number signifies releases primarily focused on new features and addressing bugs, without introducing breaking changes.
- **34** denotes the **revision number**, also referred to as the **patch number**. This number increases for weekly patch releases, which predominantly consist of bug fixes and do not introduce new features or breaking changes.

See also

- [Syncfusion product release lifecycle](#)
- [Upgrade guide](#)

Upgrading Syncfusion JavaScript (Essential JS2)

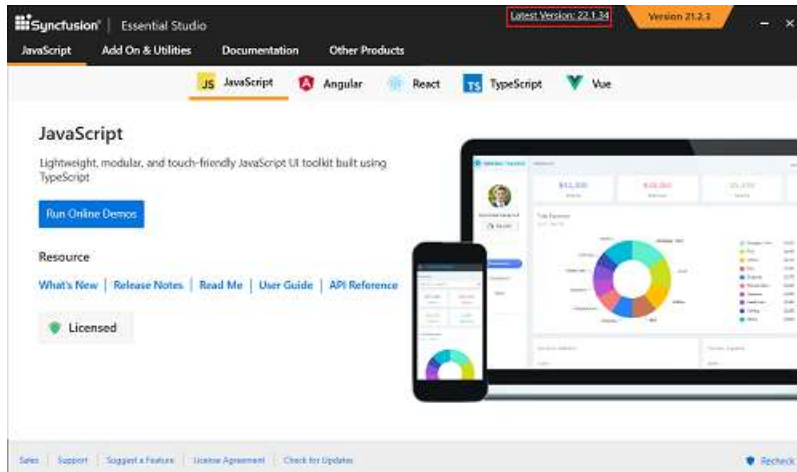
Syncfusion releases new volumes once every three months, with exciting new features. There will be one Service Pack release for these volume releases. Service Pack releases are provided to address major bug fixes in the volume releases.

You can upgrade to our latest version from any installed Syncfusion version.

See our "[Upgrade Guide](#)" for JavaScript – EJ2 to learn more about the “Breaking Changes, Bug Fixes, Features and Known Issues” between your current version and the latest version you are trying to upgrade.

Upgrading to the latest version

The most recent version of Syncfusion JavaScript – EJ2 can be downloaded and installed by clicking on the “Latest Version: {Version}” link at the top of the Syncfusion JavaScript – EJ2 Control Panel.



You can also upgrade to the latest version just by downloading and installing the products you require from [this](#) link. The existing installed versions are not required to be uninstalled.

It is not required to install the Volume release before installing the Service Pack release. As releases for Volume and Service Packs work independently, you can install the latest version with major bug fixes directly.

Upgrade from trial version to license version

Uninstall the trial version and install the fully licensed installer from the [License and Downloads](#) section of our website to upgrade from the trial version.

Note: Starting with 2022 Volume 1 v20.1.0.47, all Syncfusion customers (evaluators and paid customers) who use Syncfusion installers or npm packages must generate and register the corresponding platform and version license key in your projects.

For more information, please see the [Licensing section](#).

Licensing

Syncfusion licensing overview

We have introduced license key validation for Essential JS2 platforms from the 2022 Volume 1 release. This licensing key validation will enforce the developer to register the valid licensing key in an application while referring to any of the latest JavaScript packages, either from npm or CDN or build.

License key can be obtained from the [My Account >> License and downloads](#) section of the Syncfusion website. To obtain a license key, you will need to have an active trial or license or community license.

Before using any JavaScript controls, you must register the obtained license key in the application code. Otherwise you will get license validation error message in application as shown in below

This application was built using a trial version of Syncfusion Essential Studio. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period. Please refer to this [help topic](#) for more information.

Difference between unlock key and license key

Please note that this license key is different from the installer unlock key that you might have used in the past and needs to be separately generated from Syncfusion website.

- **Unlock Key** - Syncfusion Unlock Key is used to unlock the Syncfusion installers alone.
- **License Key** - Syncfusion License Key is a string that needs to be registered in your script to avoid licensing warning.

Refer to [this](#) KB article to know more about difference between the Syncfusion Unlock Key and the Syncfusion License Key.

Registering Syncfusion license keys in Build server

| Source of Syncfusion assemblies | Details | License Key needs to be registered? | Where to get license key from |

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

| **NuGet package** | If the Syncfusion assemblies used in Build Server were from the Syncfusion NuGet packages, then no need to install any Syncfusion installer. We can directly use the required Syncfusion NuGet packages at nuget.org.

But, if using NuGet packages from the nuget.org, then we should register the Syncfusion license key in the application. | Yes | Use any developer license to [generate](#) keys for Build Environments as well. |

| **Trial installer** | If the Syncfusion assemblies used in Build Server were from Trial Installer, we should register the license key in the application for the corresponding version and platforms, to avoid trial license warning. | Yes | Use any developer trial license to [generate](#) keys for Build Environments as well. |

| **Licensed installer** | If the Syncfusion assemblies used in Build Server were from Licensed Installer, then there is no need to register the license keys.

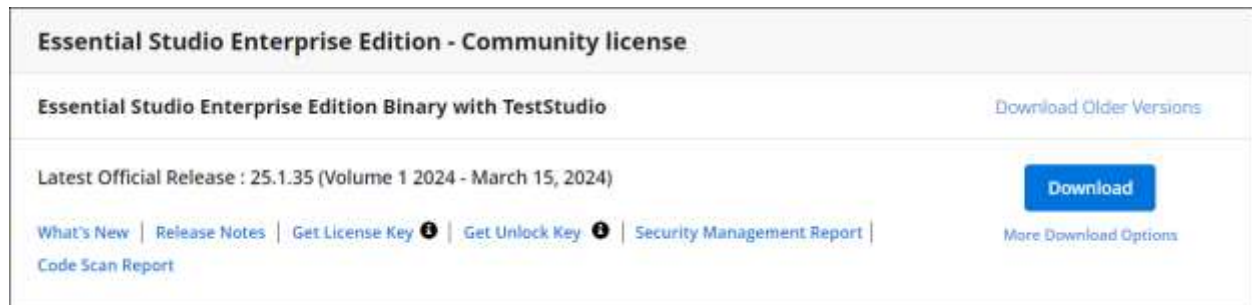
You can [download](#) and [install](#) the licensed version of our installer. | No | Not applicable |

See also

- [How to generate Syncfusion ReactJS license key?](#)
- [How to register Syncfusion license key in ReactJS application?](#)
- [Licensing FAQ](#)

Generate Syncfusion ReactJS License key

License keys can be generated from the [License & Downloads](#) or [Trial & Downloads](#) section of the Syncfusion website.



Essential Studio Enterprise Edition - Community license

Essential Studio Enterprise Edition Binary with TestStudio [Download Older Versions](#)

Latest Official Release : 25.1.35 (Volume 1 2024 - March 15, 2024)

[What's New](#) | [Release Notes](#) | [Get License Key](#) | [Get Unlock Key](#) | [Security Management Report](#) | [Code Scan Report](#)

[Download](#) [More Download Options](#)

* Syncfusion license keys are **version and platform specific**. Refer to the [KB](#) to generate the license key for the required version and platform.

* Refer to this [KB](#) to know which version of the Syncfusion license key should be used in the application.

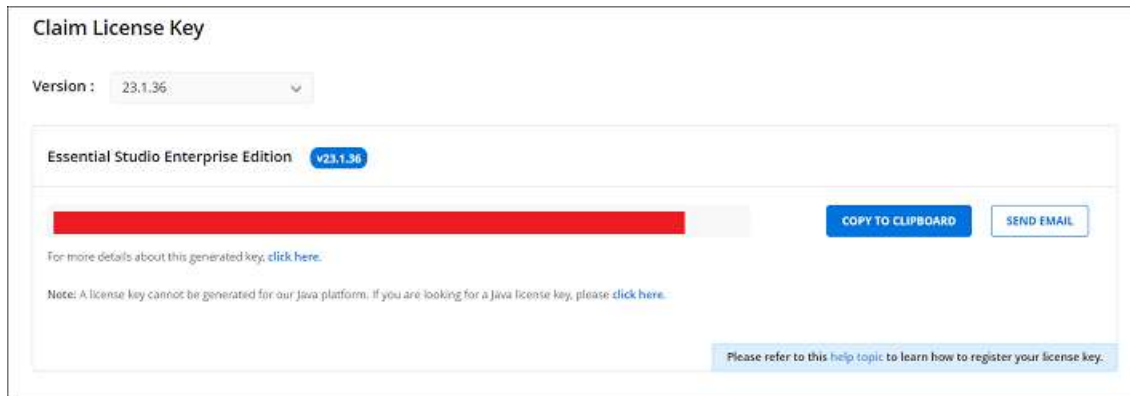
Claim license key

Syncfusion License keys can also be generated from the “**Claim License Key**” page based on the trial or valid license associated with your Syncfusion account.

You can get the license key, based on license availability in your Syncfusion account.

Active license

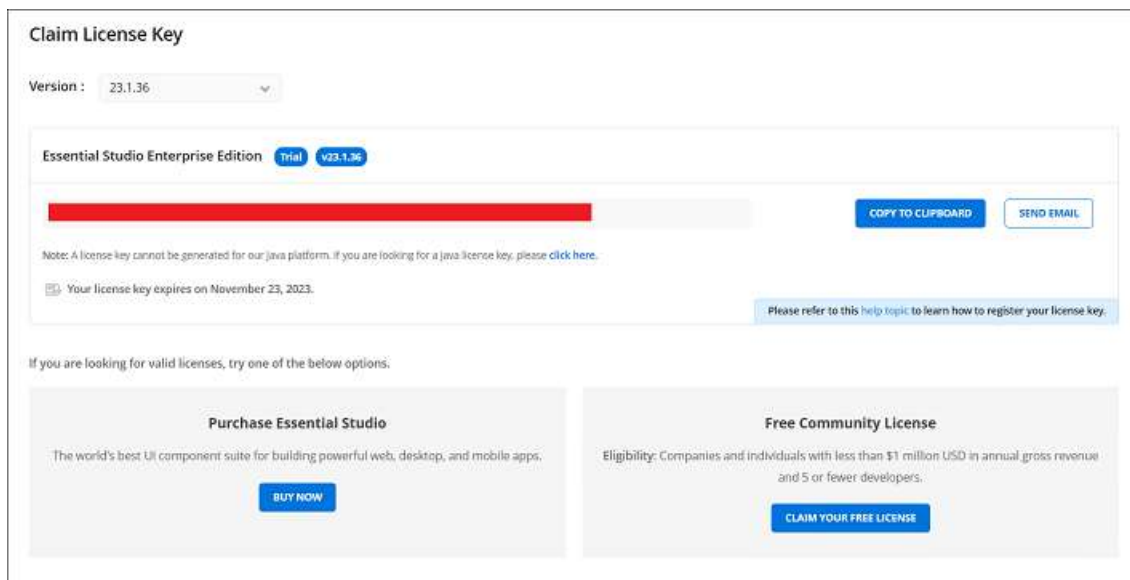
If you have a Syncfusion account associated with valid license, license key will be generated from claim license key page.



The screenshot shows the 'Claim License Key' interface. At the top, there's a 'Version' dropdown set to '23.1.36'. Below it, the license type is 'Essential Studio Enterprise Edition' with a 'v23.1.36' badge. A red bar represents the generated license key. To the right of the bar are two buttons: 'COPY TO CLIPBOARD' and 'SEND EMAIL'. Below the bar, there's a link: 'For more details about this generated key, [click here](#).' A note states: 'Note: A license key cannot be generated for our Java platform. If you are looking for a Java license key, please [click here](#).' At the bottom right, a link says: 'Please refer to this [help topic](#) to learn how to register your license key.'

Active trial

If you have a Syncfusion account associated with valid trial license, license key will be generated from claim license key page with expiry date.



This screenshot shows the 'Claim License Key' page for a trial license. It has the same top section as the active license page, but the license type is 'Essential Studio Enterprise Edition' with a 'Trial' badge and 'v23.1.36'. Below the red license key bar, there's a note: 'Note: A license key cannot be generated for our Java platform. If you are looking for a Java license key, please [click here](#).' Below that, a message says: '📅 Your license key expires on November 23, 2023.' At the bottom right, the same link is present: 'Please refer to this [help topic](#) to learn how to register your license key.'

Below this section, there's a heading: 'If you are looking for valid licenses, try one of the below options:'

Purchase Essential Studio

The world's best UI component suite for building powerful web, desktop, and mobile apps.

[BUY NOW](#)

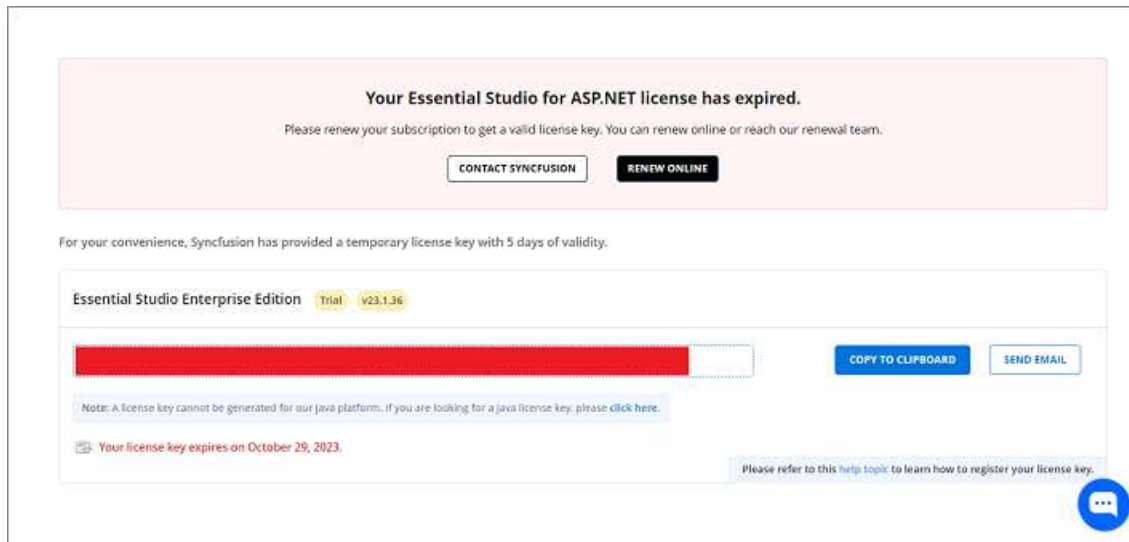
Free Community License

Eligibility: Companies and individuals with less than \$1 million USD in annual gross revenue and 5 or fewer developers.

[CLAIM YOUR FREE LICENSE](#)

Expired license

If you have a Syncfusion account with an expired license, your license subscription must be renewed in order to obtain a valid license key for the latest Essential Studio version. Meanwhile, a temporary license key with a five day validity period will be generated.



Your Essential Studio for ASP.NET license has expired.

Please renew your subscription to get a valid license key. You can renew online or reach our renewal team.

[CONTACT SYNCFUSION](#) [RENEW ONLINE](#)

For your convenience, Syncfusion has provided a temporary license key with 5 days of validity.

Essential Studio Enterprise Edition **Trial** v23.1.36

[Redacted License Key]

[COPY TO CLIPBOARD](#) [SEND EMAIL](#)

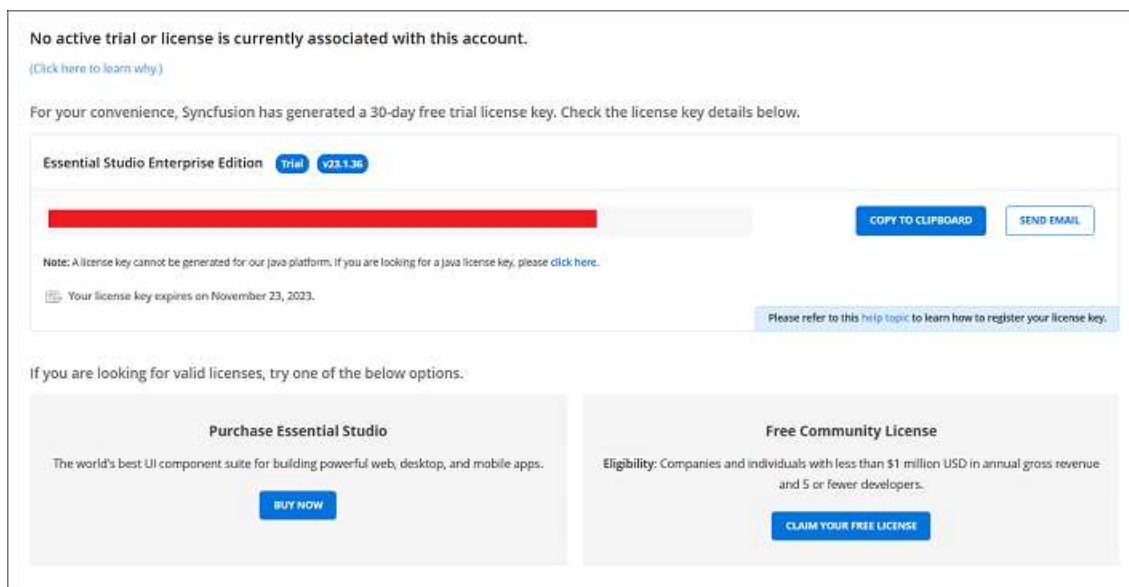
Note: A license key cannot be generated for our Java platform. If you are looking for a Java license key, please [click here](#).

Your license key expires on October 29, 2023.

Please refer to this [help topic](#) to learn how to register your license key.

No trial or no license or expired trial

If the Syncfusion account is not associated with a trial, license, or expired trial, you can try to claim either a trial or a valid license from claim license page.



No active trial or license is currently associated with this account.

[\(Click here to learn why\)](#)

For your convenience, Syncfusion has generated a 30-day free trial license key. Check the license key details below.

Essential Studio Enterprise Edition **Trial** v23.1.36

[Redacted License Key]

[COPY TO CLIPBOARD](#) [SEND EMAIL](#)

Note: A license key cannot be generated for our Java platform. If you are looking for a Java license key, please [click here](#).

Your license key expires on November 23, 2023.

Please refer to this [help topic](#) to learn how to register your license key.

If you are looking for valid licenses, try one of the below options:

Purchase Essential Studio

The world's best UI component suite for building powerful web, desktop, and mobile apps.

[BUY NOW](#)

Free Community License

Eligibility: Companies and individuals with less than \$1 million USD in annual gross revenue and 5 or fewer developers.

[CLAIM YOUR FREE LICENSE](#)

See also

- [How to register Syncfusion license key in the application?](#)
- [Licensing FAQ](#)

Register Syncfusion License key in ReactJS application

Syncfusion license key should be registered, if your project using Syncfusion ReactJS packages reference. The generated license key is a string that needs to be registered after any [Syncfusion ReactJS reference](#).

Note: Syncfusion license validation is done offline during application execution and does not require internet access. Apps registered with a Syncfusion license key can be deployed on any system that does not have an internet connection.

Generate the [Syncfusion license key](#) and register it in one of the following ways,

- [Register the license key in the project](#)
- [Register the license key using the npx command](#)

Register Syncfusion license key in the project

Register the license key in the `index.js` file of the React project.

```
`ts
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { registerLicense } from '@syncfusion/ej2-base';
// Registering Syncfusion license key
registerLicense('Replace your generated license key here');
ReactDOM.render(
  <React.StrictMode>
  <App />
</React.StrictMode>,
  document.getElementById('root')
);
`
```

Note: Only from 2022 Vol 1 v20.1.0.47, license key registration required for Essential JavaScript 2 products.

Register Syncfusion license key using the npx command

Register the Syncfusion license key through npx command in one of the following ways,

- [Register the license key with the license file](#)
- [Register the license key with the environment variable](#)

If both the license text file and the environment variable are used for license registration, priority is set to `syncfusion-license.txt` file. If you want to use the environment variable for license registration, then remove the license text file from the application.

Register the license key with the license file

The following steps show how to register the Syncfusion license key with the license text file.

- Create the `syncfusion-license.txt` file in the application root directory and paste the license key.
- Open the command prompt in the application root directory and activate the license key by using the below command,

```
npx syncfusion-license activate
```

- Once the Syncfusion license key is activated, the following console message will appear.

License message: `
` (INFO) Syncfusion License imported successfully.

- Remove the `.cache` folder from node modules in the application.
- Now run the application. If you are facing a license validation error, refer to this [link](#) to resolve it. Also, find the most frequent license registration questions from this [link](#).

If you don't want to use the license text file in the application, refer to this [link](#) to use an environment variable and register the Syncfusion license key. Also, check out some common licensing FAQs while registering the license key using the npx command from this [link](#)

Register the license key with the environment variable

You can set the environment variable as `SYNCFUSION_LICENSE` in the system and paste the license key as a value. It can be used in all applications on your machine.

The following steps show how to set environment variable in different operating systems and register the Syncfusion license key.

- Set the environment variable in different operating systems like below,

Windows

- Open the command prompt and use `setx` command to add the new environment variable.

```
setx SYNCFUSION_LICENSE "license key"
```

Mac

- Open the terminal and use the `env` command to view the variables list.
- You can set the environment variable by using below command,

```
echo 'export SYNCFUSIONLICENSE="license key"' >> ~/.bashprofile
```

- If you want to modify the environment variable in the bash profile. Use the below command,

```
nano .bash_profile
```

- Once modified the variable. Press `ctrl+x` to exit then `Y` and `Enter` button to save the changes.
- Close the terminal and open it again to see the environment variables changes using `env` command.

Linux

- Open the terminal and use the `env` command to view the variables list.
- You can set or modify the [environment variable](#) by using below command,

```
export SYNCFUSION_LICENSE='license key'
```

- Once set the `SYNCFUSION_LICENSE` environment variable, restart the IDE or application terminal before using the license activation command.
- Open the command prompt in the application root directory and activate the license key by using the below command,

```
npx syncfusion-license activate
```

- Once the Syncfusion license key is activated, the following console message will appear.

License message: `
` (INFO) Syncfusion License imported successfully.

- Remove the `.cache` folder from node modules in the application.
- Now run the application. If you are facing a license validation error, refer to this [link](#) to resolve it. Also, find the most frequent license registration questions from this [link](#).

Register Syncfusion license key in CI services

The following sections show how to use an environment variable in CI services.

GitHub actions

- Create a [new Repository Secret](#) or an [Organization Secret](#). Set the name of the secret to `SYNCFUSION_LICENSE` and use the license key as a value.
- Add the Syncfusion license activation command after running npm install or yarn like below,


```
`bash
```

```
steps:
```

- name: Install node modules

```
run: npm install
```

- name: Activate Syncfusion License

```
run: npx syncfusion-license activate
```

```
env:
```

```
SYNCFUSIONLICENSE: ${{ secrets.SYNCFUSIONLICENSE }}
```

```
,
```

Azure Pipelines (YAML)

- Create a new [User-defined Variable](#) named `SYNCFUSION_LICENSE`. Use the license key as a value.
- Add the Syncfusion license activation command after running npm install or yarn like below,

The following example shows the syntax for Windows build agents.

```
`bash
```

```
pool:
```

```
vmImage: 'windows-latest'
```

```
steps:
```

- script: call npm install

```
displayName: 'Install node modules'
```

- script: call npx syncfusion-license activate

```
displayName: 'Activate Syncfusion License'
```

```
env:
```

```
SYNCFUSIONLICENSE: $(SYNCFUSIONLICENSE)
```

```
,
```

The following example shows the syntax for Linux build agents.

```
`bash
```

```
pool:
```

```
vmImage: 'ubuntu-latest'
```

```
steps:
```

- script: npm install

displayName: 'Install node modules'

- script: npx syncfusion-license activate

displayName: 'Activate Syncfusion License'

env:

SYNCFUSIONLICENSE: \$(SYNCFUSIONLICENSE)

,

Azure Pipelines (Classic)

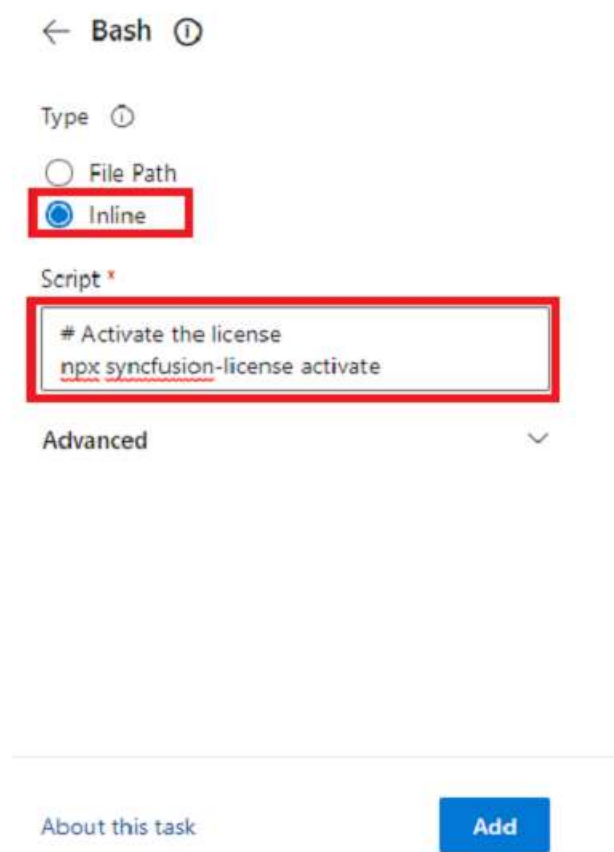
- Create a new [User-defined Variable](#) named `SYNCFUSION_LICENSE`. Use the license key as a value.
- Add the Syncfusion license activation command after running npm install or yarn using bash task like below,

`bash

Activate the license

npx syncfusion-license activate

,



The screenshot shows a configuration window for a task. At the top, there is a back arrow, the text 'Bash', and an information icon. Below this is a 'Type' section with two radio buttons: 'File Path' and 'Inline'. The 'Inline' option is selected and highlighted with a red rectangle. Underneath is a 'Script' section with a red asterisk, containing a text area with the following content: `# Activate the license` and `npx syncfusion-license activate`. This text area is also highlighted with a red rectangle. At the bottom left of the configuration area is the word 'Advanced' with a downward arrow. Below the configuration area is a horizontal line. At the bottom left, there is a link 'About this task'. At the bottom right, there is a blue button labeled 'Add'.

See also

- [Licensing FAQ](#)

Syncfusion Licensing Errors

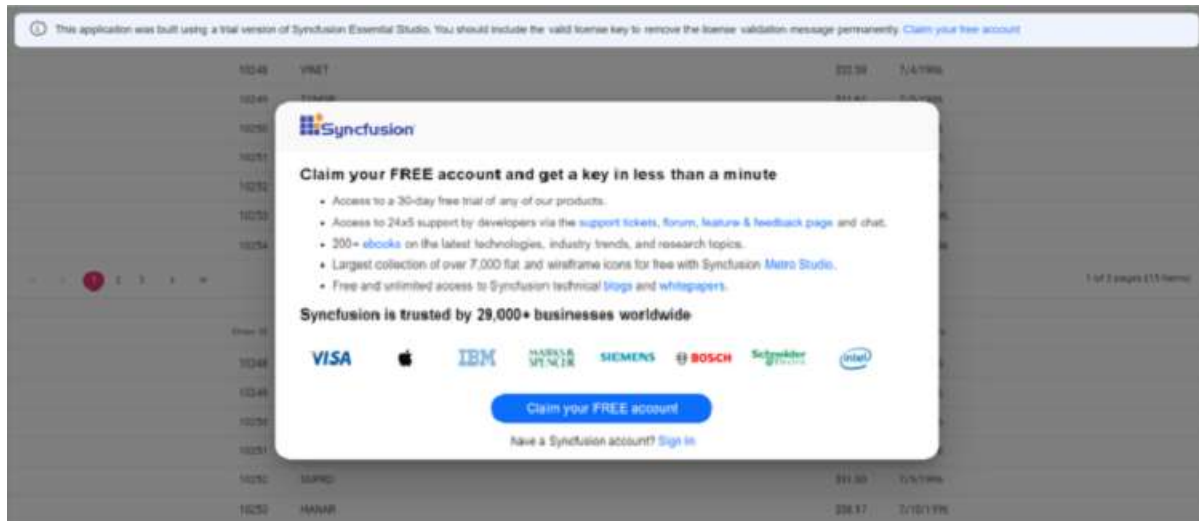
Licensing error popup is displayed with various messages under different circumstances. Here are some ways to resolve different issues.

Licensing errors

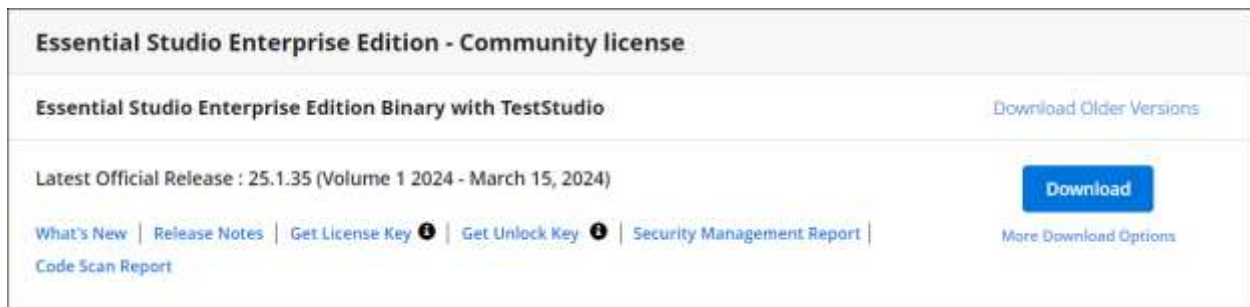
License key not registered\trial expired

The following error message will be shown if a Syncfusion license key has not been registered in your application or if the trial key has expired after 30 days.

Error message : `
` This application was built using a trial version of Syncfusion Essential Studio. You should include the valid license key to remove the license validation message permanently.

**Solution:**

- If you use ReactJS components through synCFusion installer, you can choose from the options listed below
 1. If you **have a valid SynCFusion license**, you can **generate a license key for a specific version and product** from [this page](#).

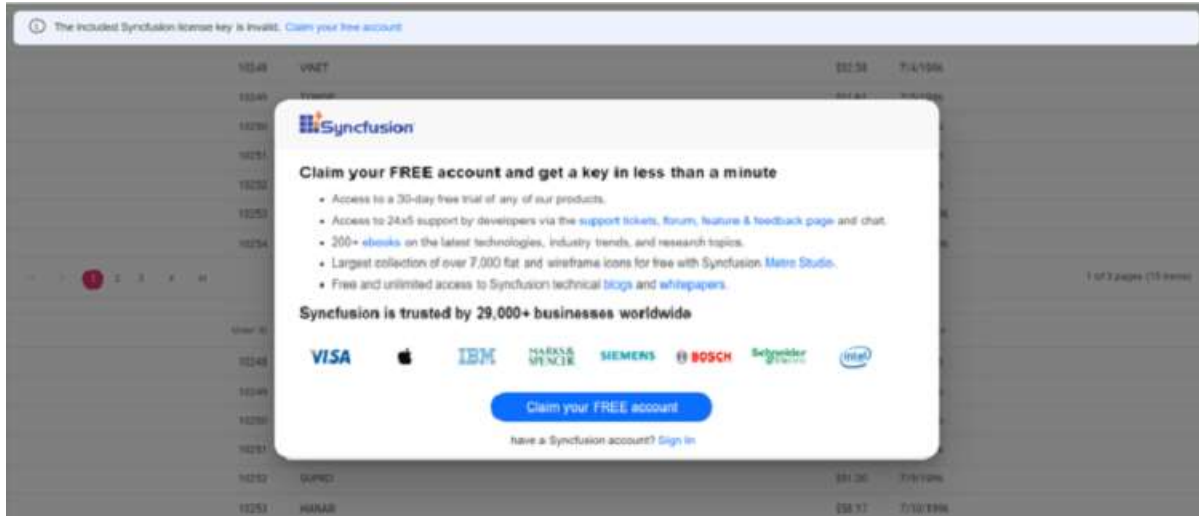


2. If you **have a SynCFusion account and an active trial**, you can **generate the trial license key for a specific version and platform** from [this page](#).
3. If you **have a SynCFusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can generate the trial license key for **a specific version and platform** from [this page](#).
4. If you **do not already have a SynCFusion account**, you can create one [here](#) and [purchase a license](#) or start your 30-day free trial. Then you can **generate the trial license key for a specific version and platform** from [this page](#).
5. Also, you can generate the license key from claim license key page by clicking the **“Claim your FREE account”** click from the licensing warning message. Refer to this [help topic](#) for more details.
 - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

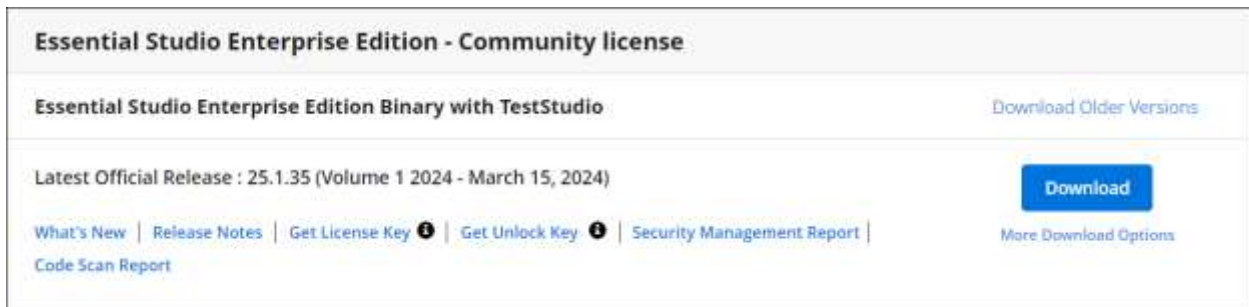
Invalid key

If the application is registered with an invalid key, another version of license key, or another platform's license key, the following error message will pop up when launching the application.

Error Message:
 The included Syncfusion license key is invalid.

**Solution:**

- If you use ReactJS components through syncfusion installer, you can choose from the options listed below
 1. If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).



2. If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
3. If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
4. If you **do not already have a Syncfusion account**, you can create one here and [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
5. Also, you can generate the license key from claim license key page by clicking the “**Claim your FREE account**” click from the licensing warning message. Refer to this [help topic](#) for more details.

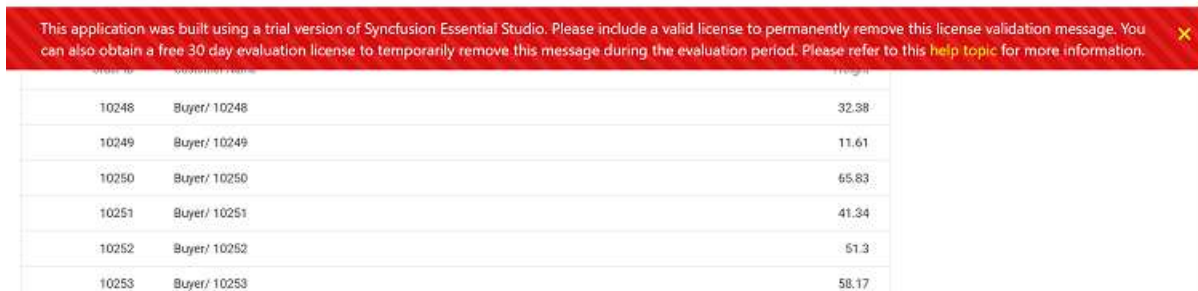
- In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

Licensing errors from version 16.2.0* to 20.3.0*

License key not registered

The following error message will be shown if a Syncfusion license key has not been registered in your application.

Error message:
 This application was built using a trial version of Syncfusion Essential Studio. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period. Please refer to this [help topic](#) for more information.



Solution:

- If you use ReactJS components through syncfusion installer, you can choose from the options listed below
 1. If you **have a valid Syncfusion license**, you can **generate a license key for a specific version and product** from [this page](#).



2. If you **have a Syncfusion account and an active trial**, you can **generate the trial license key for a specific version and platform** from [this page](#).
3. If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can generate the trial license key for **a specific version and platform** from [this page](#).
4. If you **do not already have a Syncfusion account**, you can create one [here](#) and [purchase a license](#) or start your 30-day free trial. Then you can **generate the trial license key for a specific version and platform** from [this page](#).
 - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

Invalid key

If the application is registered with an invalid key, another version of license key, or another platform's license key, the following error message will pop up when launching the application.

Error message:
 The included Syncfusion license is invalid. Please refer to this [help topic](#) for more information.



Order ID	Customer Name	Freight
10248	Buyer/ 10248	32.38
10249	Buyer/ 10249	11.61
10250	Buyer/ 10250	65.83
10251	Buyer/ 10251	41.34
10252	Buyer/ 10252	51.3
10253	Buyer/ 10253	58.17

Solution:

- If you use ReactJS components through syncfusion installer, you can choose from the options listed below
 - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).



Essential Studio Enterprise Edition - Community license

Essential Studio Enterprise Edition Binary with TestStudio [Download Older Versions](#)

Latest Official Release : 25.1.35 (Volume 1 2024 - March 15, 2024) [Download](#)

[What's New](#) | [Release Notes](#) | [Get License Key](#) | [Get Unlock Key](#) | [Security Management Report](#) | [Code Scan Report](#) [More Download Options](#)

- If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **do not already have a Syncfusion account**, you can create one here and [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
 - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

Trial expired

The following error message will be shown if the trial key has expired after 30 days.

Error message:
 Your Syncfusion trial license has expired. Please refer to this [help topic](#) for more information.

Your Syncfusion trial license has expired. Please refer to this help topic for more information.		
Order ID	Customer Name	Freight
10248	Buyer/ 10248	32.38
10249	Buyer/ 10249	11.61
10250	Buyer/ 10250	65.83
10251	Buyer/ 10251	41.34
10252	Buyer/ 10252	51.3
10253	Buyer/ 10253	58.17

Solution: Purchase from [here](#) to get a valid Syncfusion license.

Platform Mismatch

If the application is registered with another platform's license key, the following error message will pop up when launching the application.

Error message: The included Syncfusion license is invalid (Platform mismatch). Please refer to this [help topic](#) for more information.

The included Syncfusion license is invalid (Platform mismatch). Please refer to this help topic for more information.

Solution:

- License keys are version and product specific. So, if you use ReactJS components through syncfusion installer, you can choose from the options listed below
 - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).

Essential Studio Enterprise Edition - Community license

Essential Studio Enterprise Edition Binary with TestStudio [Download Older Versions](#)

Latest Official Release : 25.1.35 (Volume 1 2024 - March 15, 2024)

[What's New](#) |
 [Release Notes](#) |
 [Get License Key](#) |
 [Get Unlock Key](#) |
 [Security Management Report](#) |
 [Code Scan Report](#)

[Download](#) [More Download Options](#)

- If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
 - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

Version Mismatch

If the application is registered with another version's license key, the following error message will pop up when launching the application.

Error message:
 The included Syncfusion license ({Registered Version}) is invalid for version {Required version}. Please refer to this [help topic](#) for more information.

The included Syncfusion license (v19.1.0.44) is invalid for version 20.1.x. Please refer to this help topic for more information.		
Order ID	Customer Name	Freight
10248	Buyer/ 10248	32.36
10249	Buyer/ 10249	11.61
10250	Buyer/ 10250	65.83
10251	Buyer/ 10251	41.34
10252	Buyer/ 10252	51.3

Solution:

- License keys are version and product specific. So, if you use ReactJS components through syncfusion installer, you can choose from the options listed below
 - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).

Essential Studio Enterprise Edition - Community license

Essential Studio Enterprise Edition Binary with TestStudio [Download Older Versions](#)

Latest Official Release : 25.1.35 (Volume 1 2024 - March 15, 2024)

[What's New](#) |
 [Release Notes](#) |
 [Get License Key](#) |
 [Get Unlock Key](#) |
 [Security Management Report](#) |
 [Code Scan Report](#)

[Download](#) [More Download Options](#)

- If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
 - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

Invalid license key structure using the npx command

If you are using `npx syncfusion-license activate` command with an invalid license key structure, the following console error message will appear.

Error message:
 (Error) License key is not valid. Please refer to this [help topic](#) for more information.

Solution:

- If you use ReactJS components through syncfusion installer, you can choose from the options listed below
 - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).



2. If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
3. If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
4. If you **do not already have a Syncfusion account**, you can create one here and [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
 - In your application, register the generated license key using the npx command. Please refer to this [help topic](#) for information on registering the license key.

Licensing troubleshoot in React

Is an internet connection required for license validation

No, Internet connection is not required for the Syncfusion Essential Studio license validation. The Syncfusion license validation is done offline during application execution. Apps registered with a Syncfusion license key can be deployed on any system that does not have an internet connection.

Upgrade from the trial version after purchasing a license

To upgrade from the trial version, there are two possible solutions:

- Uninstall the trial version and install the fully licensed build from the [License & Downloads](#) section of the Syncfusion website.
- If you are using Syncfusion controls from the [npm](#), replace the currently used trial license key with a paid license key that can be generated from the [License & Downloads](#) section of Syncfusion website. Refer to [this](#) topic for more information regarding registering the license in the application.

The license registration is not required if you reference Syncfusion scripts from the Licensed installer. These licensing changes apply to all evaluators who refer to the Syncfusion scripts from the evaluation installer and those who use the npm packages from [npm](#)

Where can I get a License key

License keys can be generated from the [License & Downloads](#) or the [Trial & Downloads](#) section of the Syncfusion website.



The Syncfusion license keys are the **version and platform-specific**, refer to the [KB](#) to generate the license key for the required version and platform. Also, refer to this [KB](#) to know which version of the Syncfusion license key should be used in the application.

While using the ASP.NET Core controls with the Javascript(ES5) components, you need to register the license key in both the Javascript(ES5) and the [ASP.NET core](#). Since the license is validated at the client side for Javascript(ES5) components and server-side for the ASP.NET core components.

Will the registered license key expire

No, the Syncfusion license keys won't expire for a particular version and you can continue to use it. So, you won't face any problems on the live site. If you have used the trial key, it will expire in 30 days and we don't recommend using it in production.

If you upgrade to newer versions of the Syncfusion packages, you have to generate new license keys and use them.

When to generate new license key while upgrading

You don't have to generate and change license keys for minor version upgrades. If you upgrade from one major version and another major version, you have to generate new license keys and register in the application.

For example,

- If you upgrade to weekly releases or the SP release in the same major version, you don't have to change license such as if you upgrade from the 20.1.47 version to 20.1.* you don't have to change the license keys.
- If you upgrade from one major version to another major version, you have to generate new license keys for the latest version and change in the application, such as if you upgrade from the 20.1. version to 20.2., you have to generate new license keys for the latest version and change in the application.

License registration for multiple developers on your project

Syncfusion license key is a version based and it's not based on the developer. You don't have to register different keys for each developer. Just register one valid license key when developing and publishing the software.

Can I use the same key for all the web apps under the project

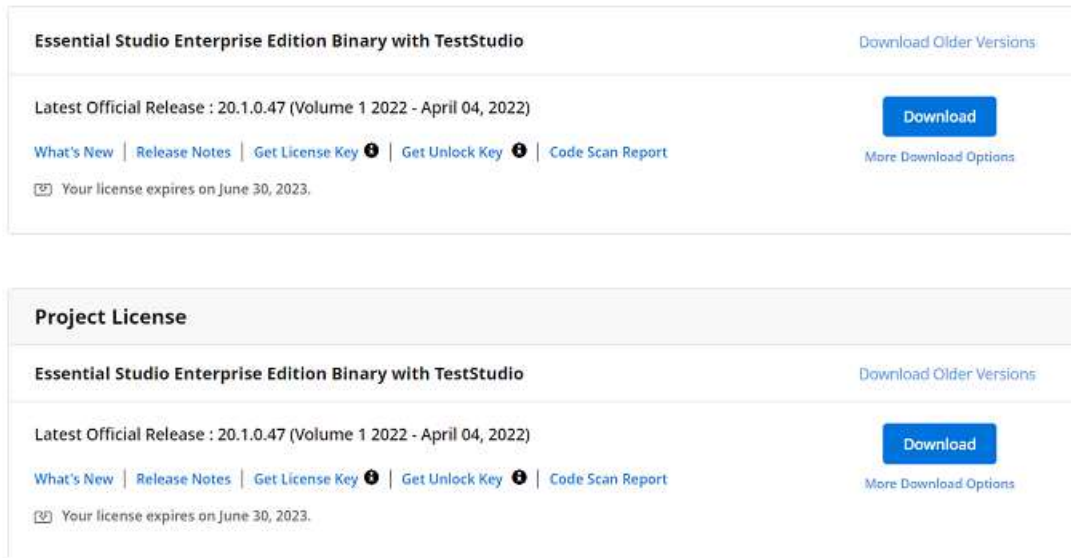
Yes, you can use the same license key for all the web apps.

Does the license registration access any resources or data

No, the license registration doesn't access any data or resources.

License & Downloads shows the "Essential Studio Enterprise Edition Binary with Test Studio" and the "Project License". Which license to use

Use any licenses shown on the [accounts & downloads](#) page. It shows two licenses because if you are part of your company's enterprise portal Global license and an individual license is also assigned to your account, on your account & downloads page, the individual license and your enterprise portal Global license are shown.



Refer to the [KB](#) article which explains the Licenses offered by Syncfusion.

If I registered the license key in both the application and the license text file
The application registered license key is set priority and used for license validation.

Reactivating license once after updating the package version while using npx

It is essential to reactivate the license key when upgrading the Syncfusion packages while the license has been registered through the `npx` command.

Potential causes of licensing errors in applications.

Below are the possible reasons that could lead to a license error within the application:

- The application may have a license issue due to duplicate Syncfusion packages.
- An invalid license issue may occur because of Syncfusion packages being referred with multiple versions.
- Registering the license key of a different version than the referred Syncfusion package version in the application can also cause licensing errors.
- Inclusion of [non-React](#) packages in the dependencies may lead to licensing errors due to the presence of duplicate instances of our packages.

License issue due to duplicate Syncfusion packages in the application

One of the possible cases on experiencing license issues in your application is due to duplicate packages exists after upgrading packages to next or latest version. To remove the duplicate packages follow the below steps.

- Delete the `@Syncfusion` folder from `node_modules` and `package-lock.json` file from app root folder.
- Clear the npm `.cache` by running the command `npm cache clean --force` or you can directly delete the file present in the application.
- It is recommended to update all Syncfusion components in the `package.json` file to the **same major version**. This ensures consistency and compatibility across the project. For instance, if the updated version being utilized is `v20.4.XX`, it is advised to upgrade all components to the **same version**.
- Run `npm install` Command.

Note: If you are using React with Next JS, ensure to remove the `.cache` created inside the `.next` folder.

Invalid license issue because of Syncfusion packages referred with multiple version

It is essential to ensure that all the components used in a project are compatible and work seamlessly together. One common issue that arises in such scenarios is **version mismatch**. Version mismatch occurs when different components have different major versions, leading to compatibility issues and difficulties in license registration.

For example, consider a situation where one component in the project has a version of `v20.1.XX`, while another component has a version of `v20.2.XX`. When such components are used together, a **version mismatch** occurs, leading to license errors. To avoid version mismatch and ensure smooth functioning of the project, it is crucial to use the **same major version** for all the Syncfusion components. This will ensure compatibility and prevent any licensing issues that may arise due to version incompatibility.

Registering the license key of a different version than the referred Syncfusion package version in the application

When developing an application with Syncfusion packages, it is important to register the appropriate license key that matches the version of the package installed. Failure to do so may result in license errors within the application.

For instance, if you are using a component version labeled as `(v20.4.XX)`, it is essential to register the license key generated **specifically** for that version. By doing so, it ensures the smooth functioning of the controls and provides access to all features and functionality without encountering any license validation errors.

License issue due to including non-React packages in the dependencies

When integrating Syncfusion with your React project, it's essential to include only our React component packages in the dependencies, as shown in the image below.

```
"@syncfusion/ej2-react-grids": "25.1.35",
"@syncfusion/ej2-react-charts": "25.1.35",
"@syncfusion/ej2-react-calendars": "25.1.35",
"@syncfusion/ej2-react-diagrams": "25.1.35",
"@syncfusion/ej2-react-documenteditor": "25.1.35",
"@syncfusion/ej2-react-pdfviewer": "25.1.35",
"@syncfusion/ej2-react-dropdowns": "25.1.35",
"@syncfusion/ej2-react-inputs": "25.1.35",
"@syncfusion/ej2-react-maps": "25.1.35",
"@syncfusion/ej2-react-pivotview": "25.1.35"
```

Avoid including our TypeScript packages separately.

```
"@syncfusion/ej2-base": "25.1.35",
"@syncfusion/ej2-icons": "25.1.35",
"@syncfusion/ej2-svg-base": "25.1.35",
"@syncfusion/ej2-pdf-export": "25.1.35",
"@syncfusion/ej2-excel-export": "25.1.35",
"@syncfusion/ej2-grids": "25.1.35",
"@syncfusion/ej2-charts": "25.1.35",
"@syncfusion/ej2-calendars": "25.1.35",
"@syncfusion/ej2-diagrams": "25.1.35",
"@syncfusion/ej2-documenteditor": "25.1.35",
"@syncfusion/ej2-pdfviewer": "25.1.35",
"@syncfusion/ej2-react-grids": "25.1.35",
"@syncfusion/ej2-react-charts": "25.1.35",
"@syncfusion/ej2-react-calendars": "25.1.35",
"@syncfusion/ej2-react-diagrams": "25.1.35",
"@syncfusion/ej2-react-documenteditor": "25.1.35",
"@syncfusion/ej2-react-pdfviewer": "25.1.35"
```

These are sub-dependencies of our React component packages and are automatically installed along with them. Including them separately may sometimes result in duplicate instances of packages, which can lead to issues with license validation. Therefore, to ensure proper license validation and avoid conflicts, stick to including our React component packages alone in your project dependencies.

Appearance

React Themes in Syncfusion Components

The Syncfusion React UI can allow you to apply styles for the components. The following list of themes are included in the Syncfusion React components library.

Theme	Style Sheet Name
-----	-----
Material 3	material3.css
Material 3 Dark	material3-dark.css

Bootstrap 5	bootstrap5.css
Bootstrap 5 Dark	bootstrap5-dark.css
Bootstrap 4	bootstrap4.css
Bootstrap 3	bootstrap.css
Bootstrap 3 Dark	bootstrap-dark.css
Google's Material	material.css
Google's Material-Dark	material-dark.css
Tailwind CSS	tailwind.css
Tailwind CSS Dark	tailwind-dark.css
Fluent	fluent.css
Fluent Dark	fluent-dark.css
Microsoft Office Fabric	fabric.css
Microsoft Office Fabric Dark	fabric-dark.css
High Contrast	highcontrast.css

The Syncfusion React Bootstrap theme is designed based on the **Bootstrap v3**, whereas the Bootstrap 4 theme is designed based on **Bootstrap v4**. In addition to these built-in themes, the [ThemeStudio](#) provides support for the Fusion Theme that can only be downloaded from the [ThemeStudio](#).

Reference themes in the React application

Using the following approaches, the themes can be referenced in the React application,

- [npm packages](#) - Used to customize the existing themes and bundle stylesheet in an application.
- [CDN](#) - Used to refer complete css via static web assest.
- [CRG](#) - Used to generate resources only for the selected (used) components.
- [Theme Studio](#) - Used to customize and generate themes only for the selected (used) components.

Instead of using the **CDN reference**, use the **npm packages** reference in your projects to customize the theme or bundle it with the other style sheets.

Refer themes through npm packages

Themes are shipped as individual and combined CSS files. The combined CSS file can be referred from the npm package **@syncfusion/ej2** and individual CSS files are available within the same component repository's **style** folder. In ej2 npm packages, we have shipped both CSS and SCSS files for all components.

To use the combined CSS files, install the npm package using the following command

```
`bash
```

```
npm install @syncfusion/ej2
```

```
,
```

Referring all components CSS

```
`css
@import "../nodemodules/@syncfusion/ej2/<themename>.css";
`
```

Referring all components SCSS

```
`
@import "../nodemodules/@syncfusion/ej2/<themename>.scss";
`
```

Referring to individual component theme

You can get the individual theme from the individual package or from ej2 package.

Referring to individual component from the **individual package**

```
`
@import "<dependent-package>/styles/<theme_name>.scss";
@import "ej2-react-buttons/styles/button/<theme_name>.scss";
`
```

Example:

```
`
@import "ej2-base/styles/material.scss";
@import "ej2-react-buttons/styles/button/material.scss";
`
```

The **ej2-base** is a common dependent package for all Syncfusion React component styles. So, it needs to be added first in the import statement.

Referring to individual component from the **ej2 package**

```
`
@import "ej2/<dependent-component>/<theme_name>.scss";
@import "ej2/button/<theme_name>.scss";
`
```

Example:

```
`
@import "ej2/base/material.scss";
@import "ej2/button/material.scss";
`
```

Advantages of individual components theme

- Reducing the page load time of application

- Reducing bundling size
- Avoid unused CSS

Refer themes through CDN reference

Instead of using a local resource on your server, use a cloud CDN to reference the theme style sheets.

Syncfusion React Themes are available in the CDN. Make sure that the version in the URLs matches the version of the Syncfusion React package you are using.

`

```
<head>
```

```
<link href="https://cdn.syncfusion.com/ej2/<version>/<theme_name>.css" rel="stylesheet"/>
```

```
</head>
```

`

Theme Name	CDN Reference
Material 3	https://cdn.syncfusion.com/ej2/22.1.34/material3.css
Material 3 Dark	https://cdn.syncfusion.com/ej2/22.1.34/material3-dark.css
Fluent	https://cdn.syncfusion.com/ej2/22.1.34/fluent.css
Fluent Dark	https://cdn.syncfusion.com/ej2/22.1.34/fluent-dark.css
Bootstrap 5	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap5.css
Bootstrap 5 Dark	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap5-dark.css
Bootstrap 4	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap4.css
Bootstrap 3	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap.css
Bootstrap 3 Dark	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap-dark.css
Google's Material	https://cdn.syncfusion.com/ej2/22.1.34/material.css
Google's Material Dark	https://cdn.syncfusion.com/ej2/22.1.34/material-dark.css
Tailwind CSS	https://cdn.syncfusion.com/ej2/22.1.34/tailwind.css
Tailwind CSS Dark	https://cdn.syncfusion.com/ej2/22.1.34/tailwind-dark.css
Microsoft Office Fabric	https://cdn.syncfusion.com/ej2/22.1.34/fabric.css
Microsoft Office Fabric Dark	https://cdn.syncfusion.com/ej2/22.1.34/fabric-dark.css
High Contrast	https://cdn.syncfusion.com/ej2/22.1.34/highcontrast.css

Theme Name	CDN Reference
Material 3	https://cdn.syncfusion.com/ej2/22.1.34/material3.css
Material 3 Dark	https://cdn.syncfusion.com/ej2/22.1.34/material3-dark.css
Fluent	https://cdn.syncfusion.com/ej2/22.1.34/fluent.css
Fluent Dark	https://cdn.syncfusion.com/ej2/22.1.34/fluent-dark.css
Bootstrap 5	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap5.css
Bootstrap 5 Dark	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap5-dark.css
Bootstrap 4	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap4.css
Bootstrap 3	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap.css
Bootstrap 3 Dark	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap-dark.css
Google's Material	https://cdn.syncfusion.com/ej2/22.1.34/material.css
Google's Material Dark	https://cdn.syncfusion.com/ej2/22.1.34/material-dark.css
Tailwind CSS	https://cdn.syncfusion.com/ej2/22.1.34/tailwind.css
Tailwind CSS Dark	https://cdn.syncfusion.com/ej2/22.1.34/tailwind-dark.css
Microsoft Office Fabric	https://cdn.syncfusion.com/ej2/22.1.34/fabric.css
Microsoft Office Fabric Dark	https://cdn.syncfusion.com/ej2/22.1.34/fabric-dark.css
High Contrast	https://cdn.syncfusion.com/ej2/22.1.34/highcontrast.css

Common variables

The following list of common variables are used in the Syncfusion JavaScript library themes for all UI controls. You can change these variables to customize the corresponding theme.

```
<!-- markdownlint-disable MD033 -->
```

Syncfusion Material 3 theme

Name	Value (Default Theme)	Value (Dark Theme)
--color-sf-black	rgb(0,0,0)	rgb(0,0,0)
--color-sf-white	rgb(255,255,255)	rgb(255,255,255)
--color-sf-primary	rgb(103, 80, 164)	rgb(208, 188, 255)
--color-sf-primary-container	rgb(234, 221, 255)	rgb(79, 55, 139)
--color-sf-on-primary	rgb(255, 255, 255)	rgb(55, 30, 115)
--color-sf-on-primary-container	rgb(33, 0, 94)	rgb(234, 221, 255)
--color-sf-surface	rgb(255, 255, 255)	rgb(28, 27, 31)
--color-sf-surface-variant	rgb(231, 224, 236)	rgb(73, 69, 79)
--color-sf-on-surface	rgb(28, 27, 31)	rgb(230, 225, 229)
--color-sf-on-surface-variant	rgb(73, 69, 78)	rgb(202, 196, 208)
--color-sf-secondary	rgb(98, 91, 113)	rgb(204, 194, 220)
--color-sf-secondary-container	rgb(232, 222, 248)	rgb(74, 68, 88)
--color-sf-on-secondary	rgb(255, 255, 255)	rgb(51, 45, 65)
--color-sf-on-secondary-container	rgb(30, 25, 43)	rgb(232, 222, 248)
--color-sf-tertiary	rgb(125, 82, 96)	rgb(239, 184, 200)
--color-sf-tertiary-container	rgb(255, 216, 228)	rgb(99, 59, 72)
--color-sf-on-tertiary	rgb(255, 255, 255)	rgb(73, 37, 50)
--color-sf-on-tertiary-containe	rgb(55, 11, 30)	rgb(255, 216, 228)
--color-sf-background	rgb(255, 255, 255)	rgb(28, 27, 31)
--color-sf-on-background	rgb(28, 27, 31)	rgb(230, 225, 229)
--color-sf-outline	rgb(121, 116, 126)	rgb(147, 143, 153)

Name	Value (Default Theme)	Value (Dark Theme)
--color-sf-outline-variant	rgb(196, 199, 197)	rgb(68, 71, 70)
--color-sf-shadow	rgb(0, 0, 0)	rgb(0, 0, 0)
--color-sf-surface-tint-color	rgb(103, 80, 164)	rgb(208, 188, 255)
--color-sf-inverse-surface	rgb(49, 48, 51)	rgb(230, 225, 229)
--color-sf-inverse-on-surface	rgb(244, 239, 244)	rgb(49, 48, 51)
--color-sf-inverse-primary	rgb(208, 188, 255)	rgb(103, 80, 164)
--color-sf-scrim	rgb(0, 0, 0)	rgb(0, 0, 0)
--color-sf-error	rgb(179, 38, 30)	rgb(242, 184, 181)
--color-sf-error-container	rgb(249, 222, 220)	rgb(140, 29, 24)
--color-sf-on-error	rgb(255, 250, 250)	rgb(96, 20, 16)
--color-sf-on-error-container	rgb(65, 14, 11)	rgb(249, 222, 220)
--color-sf-success	rgb(32, 81, 7)	rgb(83, 202, 23)
--color-sf-success-container	rgb(209, 255, 186)	rgb(22, 62, 2)
--color-sf-on-success	rgb(244, 255, 239)	rgb(13, 39, 0)
--color-sf-on-success-container	rgb(13, 39, 0)	rgb(183, 250, 150)
--color-sf-info	rgb(1, 87, 155)	rgb(71, 172, 251)
--color-sf-info-container	rgb(233, 245, 255)	rgb(0, 67, 120)
--color-sf-on-info	rgb(250, 253, 255)	rgb(0, 51, 91)
--color-sf-on-info-container	rgb(0, 51, 91)	rgb(173, 219, 255)
--color-sf-warning	rgb(145, 76, 0)	rgb(245, 180, 130)
--color-sf-warning-container	rgb(254, 236, 222)	rgb(123, 65, 0)
--color-sf-on-warning	rgb(255, 255, 255)	rgb(99, 52, 0)

Name	Value (Default Theme)	Value (Dark Theme)
--color-sf-on-warning-container	rgb(47, 21, 0)	rgb(255, 220, 193)

Syncfusion Bootstrap 5 theme

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000
\$white	#fff	#fff
\$gray-100	#f8f9fa	#f8f9fa
\$gray-200	#e9ecef	#e9ecef
\$gray-300	#dee2e6	#dee2e6
\$gray-400	#ced4da	#ced4da
\$gray-500	#adb5bd	#adb5bd
\$gray-600	#6c757d	#6c757d
\$gray-700	#495057	#495057
\$gray-800	#343a40	#343a40
\$gray-900	#212529	#212529
\$blue	#0d6efd	#0d6efd
\$indigo	#6610f2	#6610f2
\$purple	#6f42c1	#6f42c1
\$pink	#d63384	#d63384
\$red	#dc3545	#dc3545
\$orange	#fd7e14	#fd7e14
\$yellow	#ffc107	#ffc107
\$green	#198754	#198754

Name	Value (Default Theme)	Value (Dark Theme)
\$teal	#20c997	#20c997
\$cyan	#0dcaf0	#0dcaf0

Syncfusion Fluent theme

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000
\$white	#fff	#fff
\$gray220	#11100f	#11100f
\$gray210	#161514	#161514
\$gray200	#1b1a19	#1b1a19
\$gray190	#201f1e	#201f1e
\$gray180	#252423	#252423
\$gray170	#292827	#292827
\$gray160	#323130	#323130
\$gray150	#3b3a39	#3b3a39
\$gray140	#484644	#484644
\$gray130	#605e5c	#605e5c
\$gray120	#797775	#797775
\$gray110	#8a8886	#8a8886
\$gray100	#979593	#979593
\$gray90	#a19f9d	#a19f9d
\$gray80	#b3b0ad	#b3b0ad
\$gray70	#bebbb8	#bebbb8

Name	Value (Default Theme)	Value (Dark Theme)
\$gray60	#c8c6c4	#c8c6c4
\$gray50	#d2d0ce	#d2d0ce
\$gray40	#e1dfdd	#e1dfdd
\$gray30	#edebe9	#edebe9
\$gray20	#f3f2f1	#f3f2f1
\$gray10	#faf9f8	#faf9f8
\$cyanblue10	#0078d4	#0078d4
\$red10	#d13438	#d13438
\$orange20	#ca5010	#ca5010
\$green20	#0b6a0b	#0b6a0b
\$cyan20	#038387	#038387

Syncfusion Bootstrap 4 theme

Name	Value
\$white	#fff
\$gray-100	#f8f9fa
\$gray-200	#e9ecef
\$gray-300	#dee2e6
\$gray-400	#ced4da
\$gray-500	#adb5bd
\$gray-600	#6c757d
\$gray-700	#495057
\$gray-800	#343a40

Name	Value
\$gray-900	#212529
\$black	#000
\$blue	#007bff
\$indigo	#6610f2
\$purple	#6f42c1
\$pink	#e83e8c
\$red	#dc3545
\$orange	#fd7e14
\$yellow	#ffc107
\$green	#28a745
\$teal	#20c997
\$cyan	#17a2b8

Syncfusion Bootstrap theme

Name	Value (Default Theme)	Value (Dark Theme)
\$brand-primary	#317ab9	#0070f0
\$brand-primary-darken-10	#3071a9	darken(\$brand-primary, 10%)
\$brand-primary-darken-15	#2a6496	darken(\$brand-primary, 20%)
\$brand-primary-darken-25	#1f496e	darken(\$brand-primary, 30%)
\$brand-primary-darken-35	#142f46	darken(\$brand-primary, 40%)
\$brand-primary-font	#fff	#fff
\$gray-base	#000	#1a1a1a
\$gray-darker	#222	#131313

Name	Value (Default Theme)	Value (Dark Theme)
\$gray-dark	#333	#2a2a2a
\$gray	#555	#313131
\$gray-light	#777	#393939
\$grey-44	#444	#414141
\$grey-88	#888	#484848
\$grey-99	#999	#505050
\$grey-8c	#8c8c8c	#585858
\$grey-ad	#adadad	#676767
\$grey-dark-font	#fff	#f0f0f0
\$grey-white	#fff	#6e6e6e
\$grey-lighter	#eee	#767676
\$grey-f9	#f9f9f9	#7e7e7e
\$grey-f8	#f8f8f8	#858585
\$grey-f5	#f5f5f5	#8d8d8d
\$grey-e6	#e6e6e6	#959595
\$grey-dd	#ddd	#9c9c9c
\$grey-d4	#d4d4d4	#a4a4a4
\$grey-cc	#ccc	#acacac
\$grey-light-font	#333	#fff
\$brand-success	#5cb85c	#48b14c
\$brand-success-dark	#3c763d	#358238
\$brand-info	#5bc0de	#2aaac0

Name	Value (Default Theme)	Value (Dark Theme)
\$brand-info-dark	#31708f	#208090
\$brand-warning	#f0ad4e	#fac168
\$brand-warning-dark	#8a6d3b	#f9ad37
\$brand-danger	#d9534f	#d44f4f
\$brand-danger-dark	#a94442	#c12f2f
\$brand-success-light	#dff0d8	#dff0d8
\$brand-info-light	#d9edf7	#d9edf7
\$brand-warning-light	#fcf8e3	#fcf8e3
\$brand-danger-light	#f2dede	#f2dede
\$input-border-focus	#66afe9	#104888
\$brand-success-font	#3c763d	#2f7432
\$brand-info-font	#31708f	#1a6c7a
\$brand-warning-font	#8a6d3b	#9d6106
\$brand-danger-font	#a94442	#ac2a2a
\$base-font	#000	#000
\$brand-primary-lighten-10		lighten(\$brand-primary, 10%)
\$brand-primary-lighten-15		lighten(\$brand-primary, 15%)
\$brand-primary-lighten-20		lighten(\$brand-primary, 20%)
\$brand-primary-lighten-30		lighten(\$brand-primary, 30%)
\$brand-primary-lighten-40		lighten(\$brand-primary, 40%)

Syncfusion Material theme

Name	Value (Default Theme)	Value (Dark Theme)
\$accent	#e3165b	#ff80ab
\$accent-font	#fff	#000
\$primary	#3f51b5	#3f51b5
\$primary-50	#e8eaf6	#e8eaf6
\$primary-100	#c5cae9	#c5cae9
\$primary-200	#9fa8da	#9fa8da
\$primary-300	#7986cb	#7986cb
\$primary-font	#fff	#fff
\$primary-50-font	#000	#000
\$primary-100-font	#000	#000
\$primary-200-font	#000	#000
\$primary-300-font	#fff	#fff
\$grey-white	#fff	#fff
\$grey-black	#000	#000
\$grey-50	#fafafa	#fafafa
\$grey-100	#f5f5f5	#f5f5f5
\$grey-200	#eee	#eee
\$grey-300	#e0e0e0	#e0e0e0
\$grey-400	#bdbdbd	#bdbdbd
\$grey-500	#9e9e9e	#9e9e9e
\$grey-600	#757575	#757575

Name	Value (Default Theme)	Value (Dark Theme)
\$grey-700	#616161	#616161
\$grey-800	#424242	#424242
\$grey-900	#212121	#212121
\$grey-dark	#303030	#303030
\$grey-light-font	#000	#000
\$grey-dark-font	#fff	#fff
\$base-font	#000	#000
\$error-font	#f44336	#ff6652
\$success-bg		#4caf50
\$error-bg		#ff6652
\$warning-bg		#ff9800
\$info-bg		#03a9f4
\$message-font		#fff
\$success-font		#4caf50
\$warning-font		#ff9800
\$info-font		#03a9f4

Syncfusion Microsoft Office Fabric theme

Name	Value (Default Theme)	Value (Dark Theme)
\$theme-primary	#0078d6	#0074cc
\$theme-dark-alt	darken(\$theme-primary, 3%)	darken(\$theme-primary, 3%)
\$theme-dark	darken(\$theme-primary, 10%)	darken(\$theme-primary, 6%)
\$theme-darker	darken(\$theme-primary, 18%)	darken(\$theme-primary, 10%)

Name	Value (Default Theme)	Value (Dark Theme)
\$theme-secondary	lighten(\$theme-primary, 3%)	lighten(\$theme-primary, 3%)
\$theme-tertiary	lighten(\$theme-primary, 21%)	lighten(\$theme-primary, 21%)
\$theme-light	lighten(\$theme-primary, 44%)	lighten(\$theme-primary, 44%)
\$theme-lighter	lighten(\$theme-primary, 49%)	lighten(\$theme-primary, 49%)
\$theme-lighter-alt	lighten(\$theme-primary, 55%)	lighten(\$theme-primary, 55%)
\$neutral-white	#fff	#201f1f
\$neutral-lighter-alt	#f8f8f8	#282727
\$neutral-lighter	#f4f4f4	#333232
\$neutral-light	#eaeaea	#414040
\$neutral-quintenaryalt	#dadada	#4a4848
\$neutral-quintenary	#d0d0d0	#514f4f
\$neutral-tertiary-alt	#c8c8c8	#6f6c6c
\$neutral-tertiary	#a6a6a6	#9a9a9a
\$neutral-secondary-alt	#767676	#c8c8c8
\$neutral-secondary	#666	#dadada
\$neutral-primary	#333	#fff
\$neutral-dark	#212121	#f4f4f4
\$neutral-black	#000	#f8f8f8
\$alert-bg	#deecf9	#bf7500
\$error-bg	#fde7e9	#cd2a19
\$success-bg	#dff6dd	#37844d
\$theme-dark-font	#fff	#fff

Name	Value (Default Theme)	Value (Dark Theme)
\$theme-primary-font	#fff	#fff
\$theme-light-font	#333	#000
\$neutral-light-font	#333	#dadada
\$neutral-light-fontalt	#000	#fff
\$grey-dark-font	#fff	#000
\$base-font	#333	#dadada
\$message-font	#333	#fff
\$alert-font	#d83b01	#ff9d48
\$error-font	#a80000	#ff5f5f
\$success-font	#107c10	#8eff8d
\$info-bg		#1e79cb
\$info-font		#62cfff

Syncfusion High Contrast theme

Name	Value
\$selection-bg	#ffd939
\$selection-font	#000
\$selection-border	#ffd939
\$hover-bg	#685708
\$hover-font	#fff
\$hover-border	#fff
\$border-default	#969696
\$border-alt	#757575

Name	Value
\$border-fg	#fff
\$border-fg-alt	#ffd939
\$bg-base-0	#000
\$bg-base-5	#0d0d0d
\$bg-base-10	#1a1a1a
\$bg-base-15	#262626
\$bg-base-20	#333
\$bg-base-75	#bfbfbf
\$bg-base-100	#fff
\$header-font	#ffd939
\$header-font-alt	#fff
\$content-font	#fff
\$content-font-alt	#969696
\$link	#8a8aff
\$invert-font	#000
\$success-bg	#166600
\$error-bg	#b30900
\$message-font	#fff
\$alert-bg	#944000
\$info-bg	#0056b3
\$success-alt	#2bc700
\$error-alt	#ff6161

Name	Value
\$alert-alt	#ff7d1a
\$info-alt	#66b0ff
\$disable	#757575

Syncfusion Tailwind CSS theme

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000
\$white	#fff	#fff
\$transparent	transparent	transparent
\$cool-gray-50	#f9fafb	#f9fafb
\$cool-gray-100	#f3f4f6	#f3f4f6
\$cool-gray-200	#e5e7eb	#e5e7eb
\$cool-gray-300	#d1d5db	#d1d5db
\$cool-gray-400	#9ca3af	#9ca3af
\$cool-gray-500	#6b7280	#6b7280
\$cool-gray-600	#4b5563	#4b5563
\$cool-gray-700	#374151	#374151
\$cool-gray-800	#1f2937	#1f2937
\$cool-gray-900	#111827	#111827
\$red-100	#fee2e2	#fee2e2
\$red-400	#f87171	#f87171
\$red-500	#ef4444	#ef4444
\$red-600	#dc2626	#dc2626

Name	Value (Default Theme)	Value (Dark Theme)
\$red-800	#991b1b	#991b1b
\$green-100	#dcfce7	#dcfce7
\$green-500	#22c55e	#22c55e
\$green-600	#16a34a	#16a34a
\$green-700	#15803d	#15803d
\$orange-100	#ffedd5	#ffedd5
\$orange-500	#f97316	#f97316
\$orange-600	#ea580c	#ea580c
\$orange-700	#c2410c	#c2410c
\$orange-800	#9a3412	#9a3412
\$cyan-300	#67e8f9	#67e8f9
\$cyan-400	#22d3ee	#22d3ee
\$cyan-500	#06b6d4	#06b6d4
\$cyan-600	#0891b2	#0891b2
\$cyan-800	#155e75	#155e75
\$indigo-50	#eef2ff	
\$indigo-100	#e0e7ff	
\$indigo-200	#c7d2fe	
\$indigo-300	#a5b4fc	
\$indigo-400	#818cf8	
\$indigo-500	#6366f1	
\$indigo-600	#4f46e5	

Name	Value (Default Theme)	Value (Dark Theme)
\$indigo-700	#4338ca	
\$indigo-800	#3730a3	
\$indigo-900	#312e81	
\$green-400		#4ade80
\$light-blue-50		#f0f9ff
\$light-blue-100		#e0f2fe
\$light-blue-400		#38bdf8
\$light-blue-500		#0ea5e9
\$light-blue-600		#0284c7
\$light-blue-700		#0369a1
\$light-blue-800		#075985

See also

- [Syncfusion icons and customization](#)
- [Theme Studio for Syncfusion components](#)

Size Mode for Syncfusion React Components

An application that is designed to be accessed through a web browser on various devices, including desktop computers and mobile devices, may have a distinct layout or user interface on a mobile device compared to a desktop computer to better suit the smaller screen size.

Syncfusion React components support both touch (bigger) and normal size modes. Touch mode creates a responsive design for mobile devices by adding the `e-bigger` class, which enhances interactions, visibility, and the overall experience.

Size mode for application

The user can enable touch mode (bigger) for the entire application by adding the `e-bigger` class to the `body` element in the `index.html` file as follows:

`

```
<body className="e-bigger">
```

...

```
</body>
```

Size mode for a component

The user can enable touch mode (bigger) for a component by adding the `e-bigger` class to the `div` element that contains the component. Another way of enabling touch mode is by adding the `e-bigger` class using the available `cssClass` property of the component.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
//Enable ripple effect
enableRipple(true);
function App() {
    const content = "Button";
    return (<div className="e-bigger">
        <ButtonComponent content={content}></ButtonComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
//Enable ripple effect
enableRipple(true);
function App() {
    const content = "Button";
    return (
        <div className="e-bigger">
            <ButtonComponent content={content}></ButtonComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Button</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-
calendars/styles/material.css" rel="stylesheet" />
<link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
grids/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://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='sample'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Change size mode for application at runtime

The user can change the size mode of the application between touch and normal (mouse) mode at runtime by adding and removing the `e-bigger` class. The following steps explain how to change the size mode of an application at runtime:

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { CalendarComponent } from '@syncfusion/ej2-react-calendars';
import { enableRipple } from '@syncfusion/ej2-base';
//Enable ripple effect
enableRipple(true);
function App() {
    const touchContent = "Touch Mode";
    const mouseContent = "Mouse Mode";
    function btnClick() {
        document.body.classList.add('e-bigger');
    }
    ;
    function mouseClick() {
        document.body.classList.remove('e-bigger');
    }
}

```

```

    }
    return (<div>
      <ButtonComponent id='touch' content={touchContent}
onClick={btnClick.bind(this)}></ButtonComponent>
      <ButtonComponent id='mouse' content={mouseContent}
onClick={mouseClick.bind(this)}></ButtonComponent>
      <div>
        <CalendarComponent id="calendar"/>
      </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 { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { CalendarComponent } from '@syncfusion/ej2-react-calendars';
import { enableRipple } from '@syncfusion/ej2-base';
//Enable ripple effect
enableRipple(true);
function App() {
  const touchContent = "Touch Mode";
  const mouseContent = "Mouse Mode";
  function btnClick() {
    document.body.classList.add('e-bigger');
  };
  function mouseClick() {
    document.body.classList.remove('e-bigger');
  }
  return (
    <div>
      <ButtonComponent id='touch' content={touchContent}
onClick={btnClick.bind(this)}></ButtonComponent>
      <ButtonComponent id='mouse' content={mouseContent}
onClick={mouseClick.bind(this)}></ButtonComponent>
      <div>
        <CalendarComponent id="calendar"/>
      </div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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-
calendars/styles/material.css" rel="stylesheet" />
    <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
grids/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://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='sample'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Change size mode for a component at runtime

The user can change the size mode of a component between touch and normal (mouse) mode at runtime by setting the **e-bigger** CSS class.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { CalendarComponent } from '@syncfusion/ej2-react-calendars';
import { enableRipple } from '@syncfusion/ej2-base';
//Enable ripple effect
enableRipple(true);
function App() {
    const touchContent = "Touch Mode";
    const mouseContent = "Mouse Mode";
    function btnClick() {
        let controls = document.querySelectorAll('.control');
        for (let index = 0; index < controls.length; index++) {
            controls[index].classList.add('e-bigger');
        }
    }
}

```

```

;
function mouseClicked() {
    let controls = document.querySelectorAll('.control');
    for (let index = 0; index < controls.length; index++) {
        controls[index].classList.remove('e-bigger');
    }
}
return (<div>
    <ButtonComponent id='touch' content={touchContent}
onClick={btnClick.bind(this)}></ButtonComponent>
    <ButtonComponent id='mouse' content={mouseContent}
onClick={mouseClick.bind(this)}></ButtonComponent>
    <div className="control">
        <CalendarComponent id="calendar"/>
    </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 { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { CalendarComponent } from '@syncfusion/ej2-react-calendars';
import { enableRipple } from '@syncfusion/ej2-base';
//Enable ripple effect
enableRipple(true);
function App() {
    const touchContent = "Touch Mode";
    const mouseContent = "Mouse Mode";
    function btnClick() {
        let controls = document.querySelectorAll('.control');
        for (let index: number = 0; index < controls.length; index++) {
            controls[index].classList.add('e-bigger');
        }
    };
    function mouseClicked() {
        let controls = document.querySelectorAll('.control');
        for (let index: number = 0; index < controls.length; index++) {
            controls[index].classList.remove('e-bigger');
        }
    }
    return (
        <div>
            <ButtonComponent id='touch' content={touchContent}
onClick={btnClick.bind(this)}></ButtonComponent>
            <ButtonComponent id='mouse' content={mouseContent}
onClick={mouseClick.bind(this)}></ButtonComponent>
            <div className="control">
                <CalendarComponent id="calendar"/>
            </div>
        </div>
    );
}

```

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

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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-
calendars/styles/material.css" rel="stylesheet" />
  <link href="https://cdn.syncfusion.com/ej2/20.3.56/ej2-react-
grids/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://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='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

Change the font size for all components

The user can change the font size for all the components by overriding the CSS for the **e-control** class as follows:

```
`css
.e-control, .e-control [class^='e-'], .e-control [class*='e-'] {
font-size: 1rem;
```

```
}
`
```

Change the font family of Syncfusion React components

The font family of Syncfusion React components can be changed by using the `e-control` class. This class is present in all Syncfusion components and allows the user to change the font family of all Syncfusion React components in an application.

```
`css
.e-control {
font-family: Arial !important;
}
`
```

By including the above CSS code block in the style sheet of the application, the user can change the font-family of all Syncfusion components.

See also

- [Sidebar responsiveness](#)
- [DataGrid responsiveness](#)
- [TreeGrid responsiveness](#)
- [Dashboard Layout responsiveness](#)
- [Kanban responsiveness](#)
- [Toolbar responsiveness](#)
- [Tab responsiveness](#)

React Icons Library

Syncfusion's icon library is a collection of pre-designed icons that can be used to enhance the user interface of an application. This pre-designed icons are set of `base64` formatted font icons. Utilizing this icon library can make it simpler to create a cohesive, visually pleasing design for an application.

Referring icons in the React application

Using the below approaches, the icons can be referenced in the React application.

- [npm package](#) - Use the npm package to access icons.
- [CDN reference](#) - Use the static web asset to access icons.

The npm package

All Syncfusion theme icons are shipped in the [ej2-icons](#) package, which is published on the [npmjs.com](#) public registry. This package contains both CSS and SCSS theme files for all themes.

Icons can be used from the npm package `ej2-icons`. To use the icons, install the npm package using the following command:

```
`bash
npm install @syncfusion/ej2-icons
`
```


Refer to the following syntax to use icons in a React application:

```
`css
@import "../nodemodules/@syncfusion/ej2-icons/styles/<themenam>.css";
`
```

Example:

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

CDN reference

All Syncfusion theme icons are available on the CDN. Instead of using a local resource on the server, use a cloud CDN to refer to the icons.

Make sure that the version of the icons in the URL matches the version of the Syncfusion React package. This will prevent compatibility issues and ensure that the correct version of the icons is loaded.

To use the icons from the CDN, refer to the icons by URLs in the application. This can be done by linking the icons in the HTML file by adding a link tag to the head section.

```
`
// Bootstrap5
<head>
<link href="https://cdn.syncfusion.com/ej2/ej2-icons/styles/bootstrap5.css" rel="stylesheet"/>
</head>
//Material
<head>
<link href="https://cdn.syncfusion.com/ej2/ej2-icons/styles/material.css" rel="stylesheet"/>
</head>
`
```

Steps to use icons library

Let's create a React application using the following command:

```
`bash
npx create-react-app my-app
cd my-app
npm start
`
```

For an introduction and configuration of the common specifications, see [getting started with the Syncfusion React application](#).

Using icons directly in HTML element

The built-in Syncfusion icons can be rendered directly in the HTML element by defining the `e-icons` class, which contains the font-family and common properties of font icons, and defining the available icon's class with the `e-` prefix.

The following steps explain the direct rendering of the Syncfusion icon in the HTML element.

1. Add the class name `e-icons` to the HTML element that needs to render the icon.
2. Add the icon class with corresponding icon content from the [available icons](#). For example, the below code snippet represents the paste icon class.

```
`css
.e-paste:before {
content:'\e355';
}
`
```

3. Add `e-icons` and `e-paste` classes to the HTML element.

```
<span class="e-icons e-paste"></span>
```

4. Add the CDN link reference of icons library in the `~index.html` file.

```
<link href="https://cdn.syncfusion.com/ej2/ej2-icons/styles/bootstrap5.css" rel="stylesheet" />
```

The below code snippet represents a complete example of icon usage.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    return (<div className="icon-bar">
        <span className="e-icons e-cut"></span>
        <span className="e-icons e-copy"></span>
        <span className="e-icons e-paste"></span>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  return (
    <div className="icon-bar">
      <span className="e-icons e-cut"></span>
      <span className="e-icons e-copy"></span>
      <span className="e-icons e-paste"></span>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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="index.css" rel="stylesheet" />
  <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-
grids/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://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: #008c00;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
```

```
</html>
```

Icon size

The `ej2-icons` package offers options to display icons in different size modes. A user can use different icon sizes in their application based on touch or mouse mode. If the user is using touch mode, add `e-large` class to the element to make the icon easily interactable, or add the `e-small` or `e-medium` class in mouse mode.

The pre-defined icon size is present in the available classes listed below.

- `e-small` - Sets the icon size as 8px.
- `e-medium` - Sets the icon size to 16px.
- `e-large` - Sets the icon size to 24px.

Example:

```
,
```

```
<span class="e-icons e-small e-search"></span>
```

```
<span class="e-icons e-medium e-search"></span>
```

```
<span class="e-icons e-large e-search"></span>
```

```
,
```

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    return (
        <div>
            <p><b>Smaller Icons</b></p>
            <div className="small-icon-bar">
                <span className="e-icons e-small e-cut"></span>
                <span className="e-icons e-small e-copy"></span>
                <span className="e-icons e-small e-paste"></span>
            </div>
            <p><b>Medium Icons</b></p>
            <div className="medium-icon-bar">
                <span className="e-icons e-medium e-cut"></span>
                <span className="e-icons e-medium e-copy"></span>
                <span className="e-icons e-medium e-paste"></span>
            </div>
            <p><b>Larger Icons</b></p>
            <div className="large-icon-bar">
                <span className="e-icons e-large e-cut"></span>
                <span className="e-icons e-large e-copy"></span>
                <span className="e-icons e-large e-paste"></span>
            </div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  return (
    <div>
      <p><b>Smaller Icons</b></p>
      <div className="small-icon-bar">
        <span className="e-icons e-small e-cut"></span>
        <span className="e-icons e-small e-copy"></span>
        <span className="e-icons e-small e-paste"></span>
      </div>
      <p><b>Medium Icons</b></p>
      <div className="medium-icon-bar">
        <span className="e-icons e-medium e-cut"></span>
        <span className="e-icons e-medium e-copy"></span>
        <span className="e-icons e-medium e-paste"></span>
      </div>
      <p><b>Larger Icons</b></p>
      <div className="large-icon-bar">
        <span className="e-icons e-large e-cut"></span>
        <span className="e-icons e-large e-copy"></span>
        <span className="e-icons e-large e-paste"></span>
      </div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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="index.css" rel="stylesheet" />
  <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-
grids/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://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='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Icon appearance customization

The Syncfusion React icons can be customized with custom color and size by overriding the `e-icons` class. Customizing the icons in the library can be useful for making the icons more visually appealing and fitting to the design of the application. For example, a user can change the color of an icon to match the color scheme of their application, or increase the size of an icon to make it more visible on smaller screens. It may also be useful for creating a consistent look and feel across different parts of the application. Overall, customizing the icons in the library can improve the overall user experience of the application.

In the example below, the icon colour is customized with custom color.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {
  return (<div>
    <div className="icon-bar">
      <span className="e-icons e-cut"></span>
      <span className="e-icons e-copy"></span>
      <span className="e-icons e-paste"></span>
    </div>
  </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
function App() {

```

```

    return (
      <div>
        <div className="icon-bar">
          <span className="e-icons e-cut"></span>
          <span className="e-icons e-copy"></span>
          <span className="e-icons e-paste"></span>
        </div>
      </div>
    );
  }
  export default App;
  ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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="index.css" rel="stylesheet" />
  <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-
grids/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://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: #008c00;
      height: 40px;
      left: 45%;
      position: absolute;
      top: 45%;
      width: 30%;
    }
  </style>
</head>
<body>
  <div id='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Available icons

The complete package of Essential JS 2 icons is listed below. The corresponding icon content can be referred in the content section.

<!-- markdownlint-disable MD033 -->

Material

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/material/demo.html"
style="height:1000px;width:100%;"></iframe>
```

Fabric

```
<iframe class="doc-sample-frame" src="https://ej2.syncfusion.com/products/icons/fabric/demo.html"
style="height:1000px;width:100%;"></iframe>
```

Bootstrap

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/bootstrap/demo.html"
style="height:1000px;width:100%;"></iframe>
```

Bootstrap 4

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/bootstrap4/demo.html"
style="height:1000px;width:100%;"></iframe>
```

Bootstrap 5

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/bootstrap5/demo.html"
style="height:1000px;width:100%;"></iframe>
```

High Contrast

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/highcontrast/demo.html"
style="height:1000px;width:100%;"></iframe>
```

Tailwind CSS

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/tailwind/demo.html"
style="height:1000px;width:100%;"></iframe>
```

Fluent

```
<iframe class="doc-sample-frame" src="https://ej2.syncfusion.com/products/icons/fluent/demo.html"
style="height:1000px;width:100%;"></iframe>
```

See also

- [Using icons in Syncfusion React Button](#)

Overview

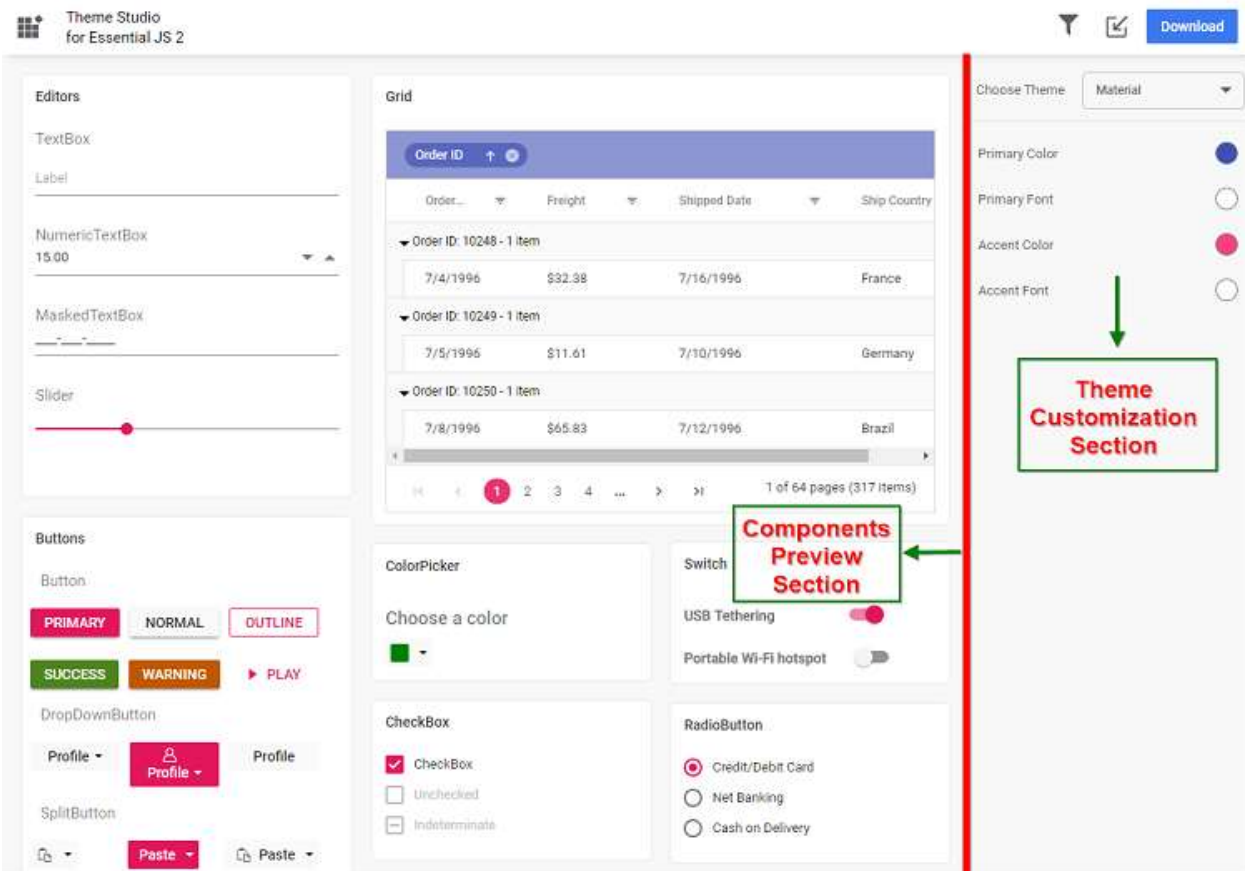
Theme Studio for Essential JS 2 can be used to customize a new theme from an existing theme. It doesn't support with Data visualization components like Chart, Diagram, Gauge, Range Navigator, Maps.

Customizing theme color from theme studio

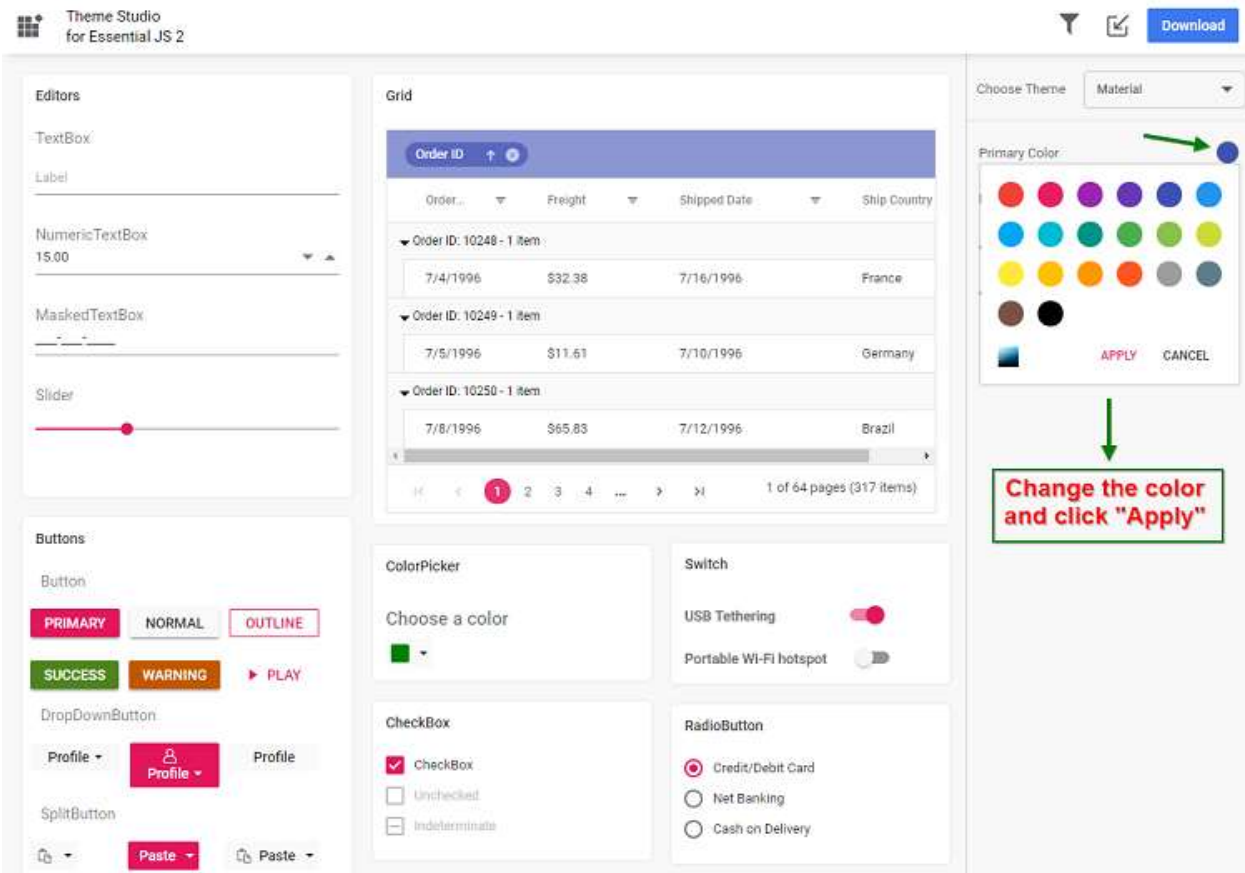
The Essential JS 2 themes are developed under the SCSS environment. Each theme has a unique common variable list. When you change the common variable color code value, it will reflect in all the Syncfusion React components.

All Syncfusion React component styles are derived from these [theme-based common variables](#). This common variable list is handled inside the theme studio application for customizing theme-based colors.

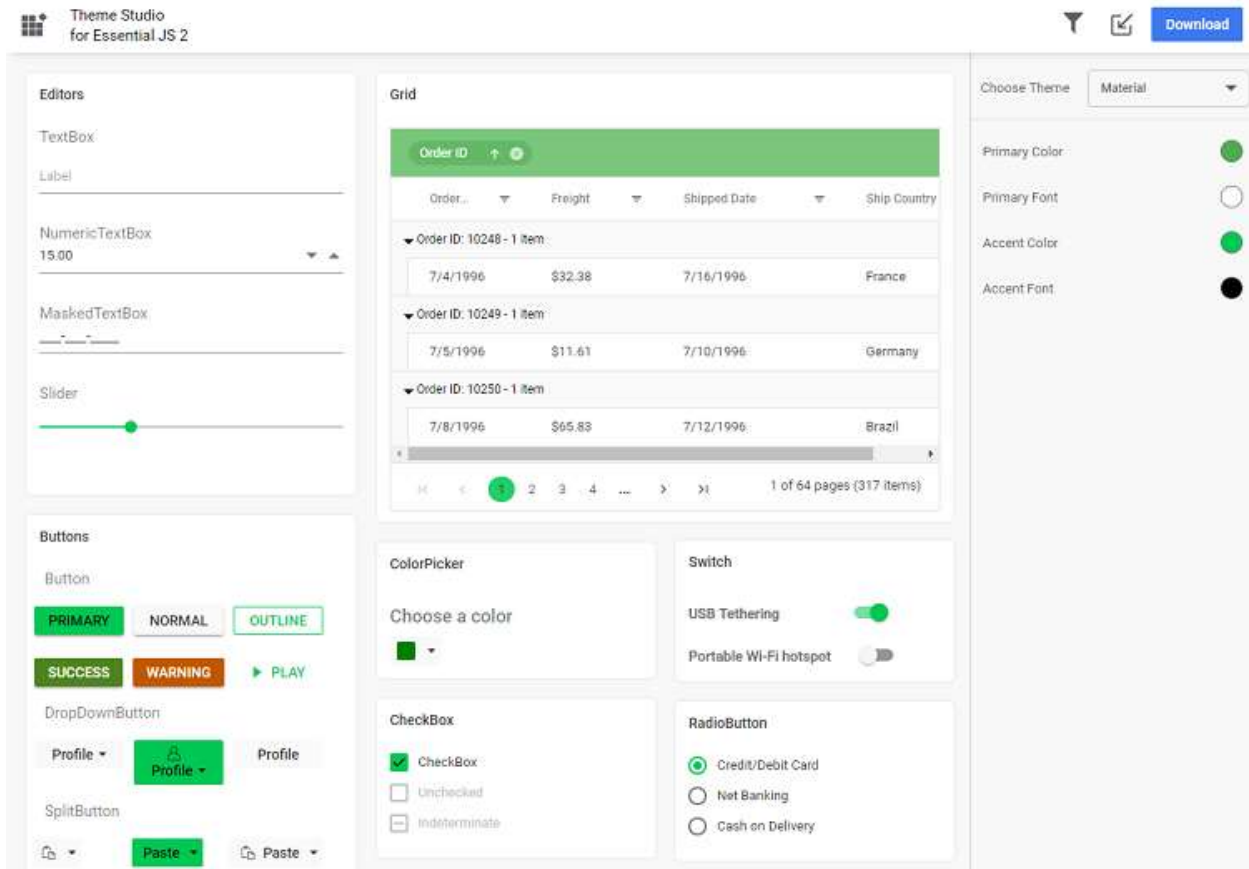
1. Navigate to the theme studio application at <https://ej2.syncfusion.com/themestudio/>.
2. The theme studio application page can be divided into two sections: the components preview section on the left, and the theme customization section on the right.



3. Click the color pickers in the theme customization section to select your desired color.



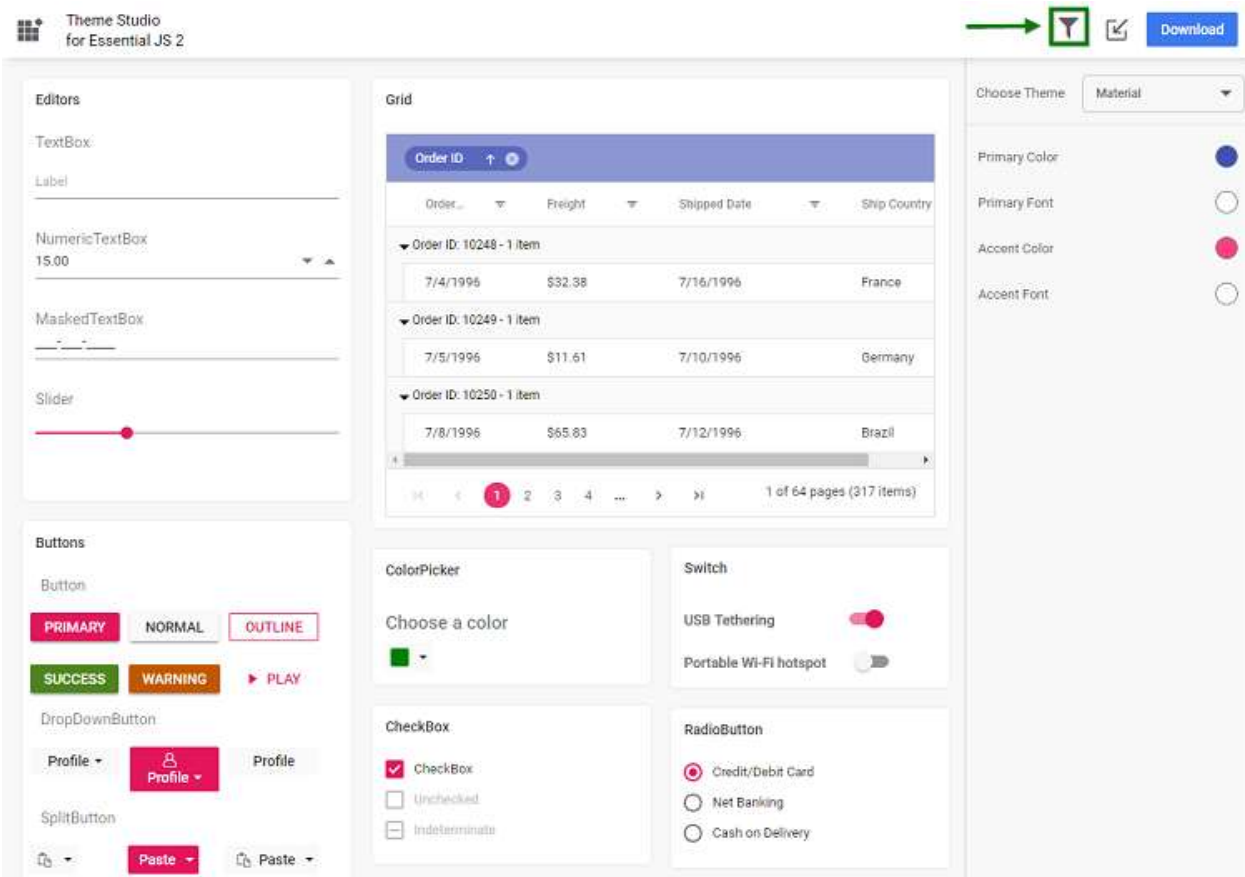
4. The Syncfusion React components will render with the newly selected color in the preview section, after selecting the custom color pickers.



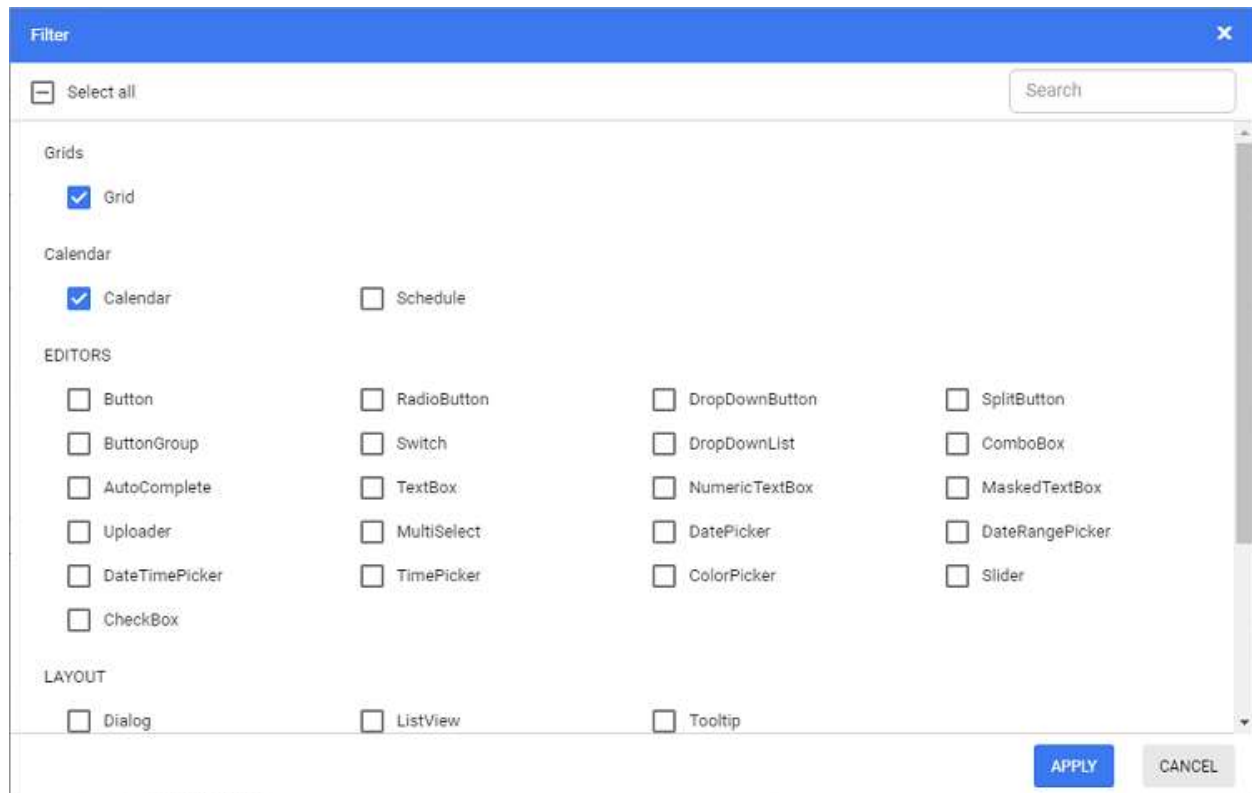
Filtering a specific list of components

Using the theme studio, you can apply custom themes to a list of specific components. This option is useful when you have integrated a selective list of Syncfusion React components in your application. The theme studio will filter the selected components and customize the final output for those components' styles alone, reducing the final output file size.

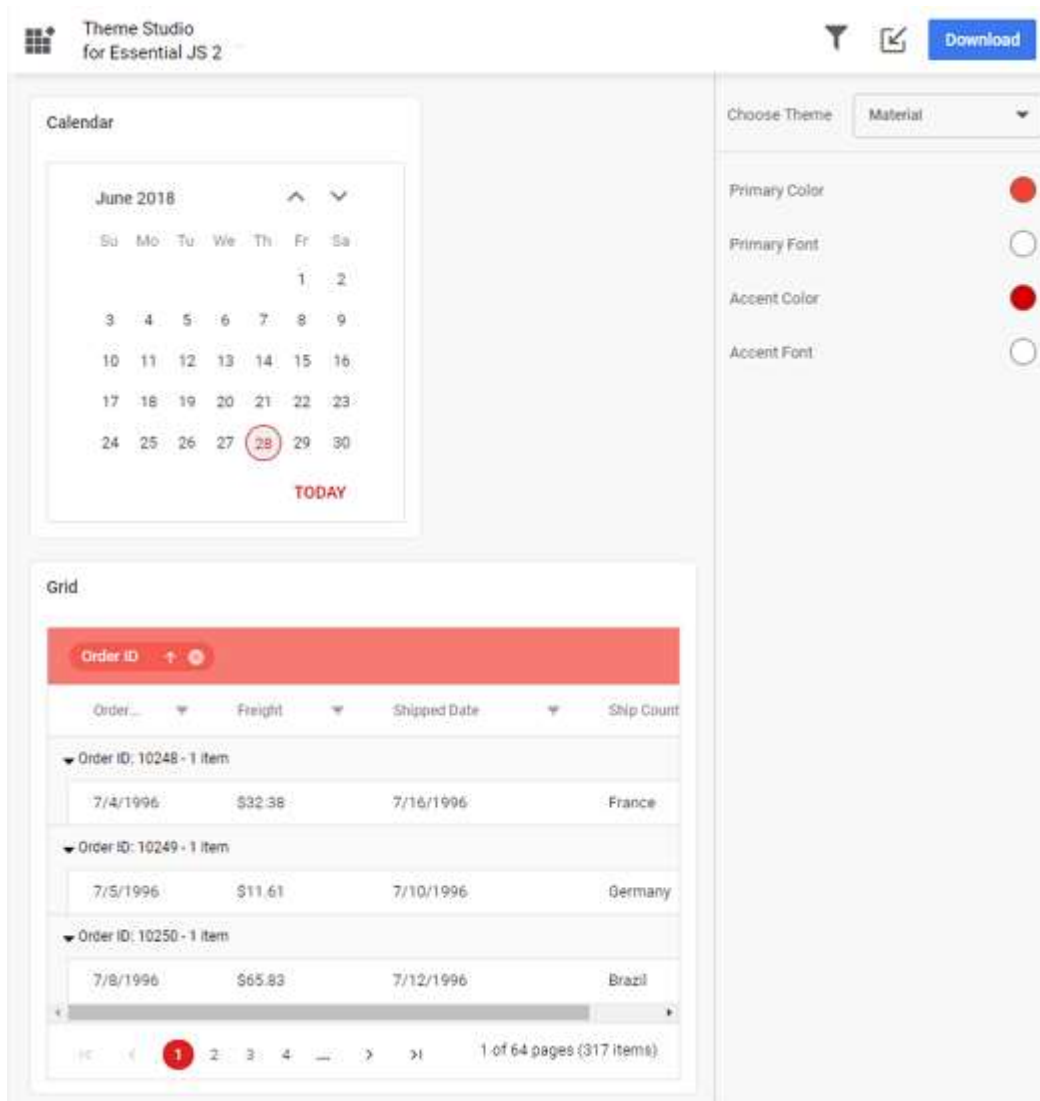
1. Click the Filter icon in the top right corner and select the components whose theme you want to customize.



2. Click the Apply button in the Filter dialog. Now, only the selected components will be rendered in the components preview section.



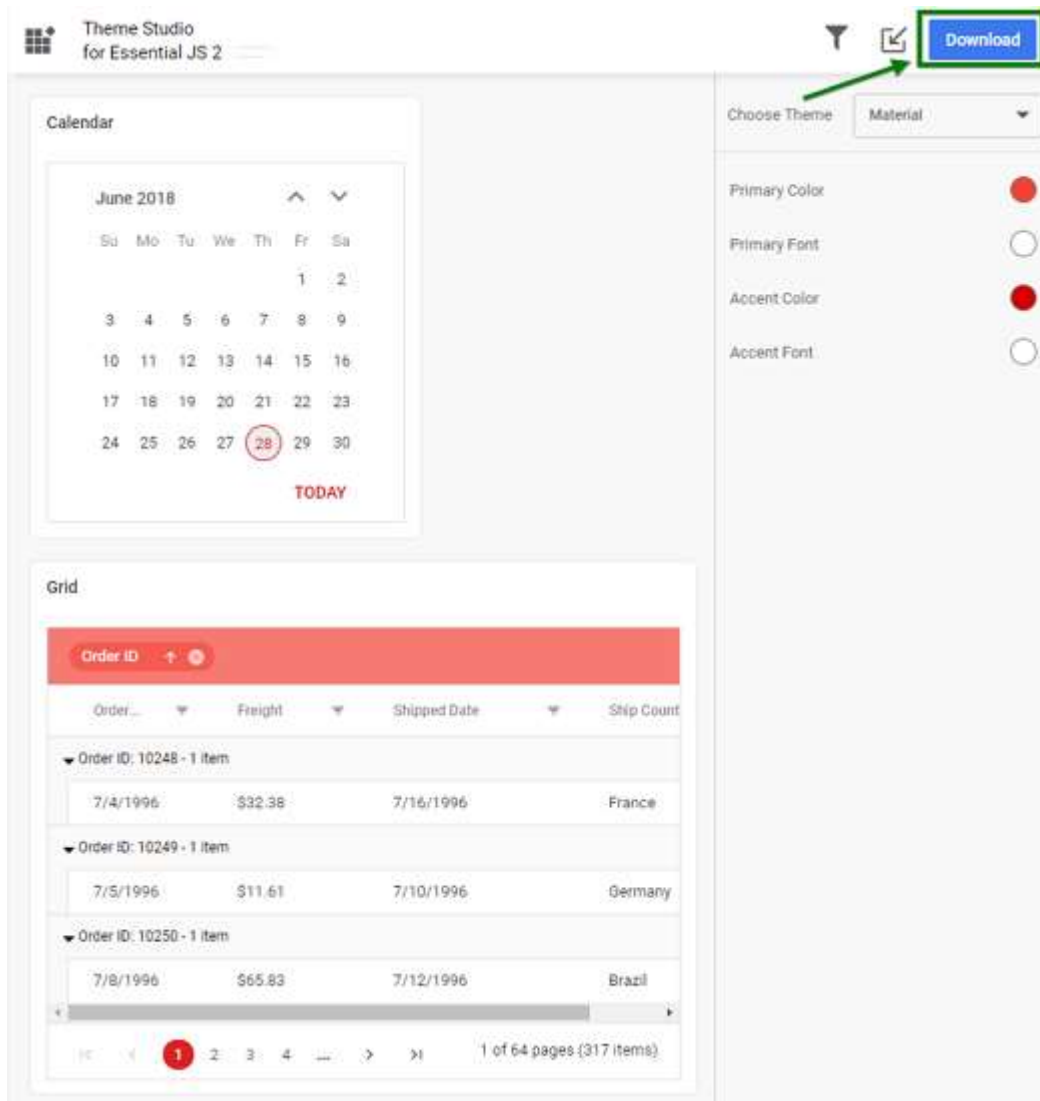
3. Now you can customize the colors in the theme customization section for the components you selected.



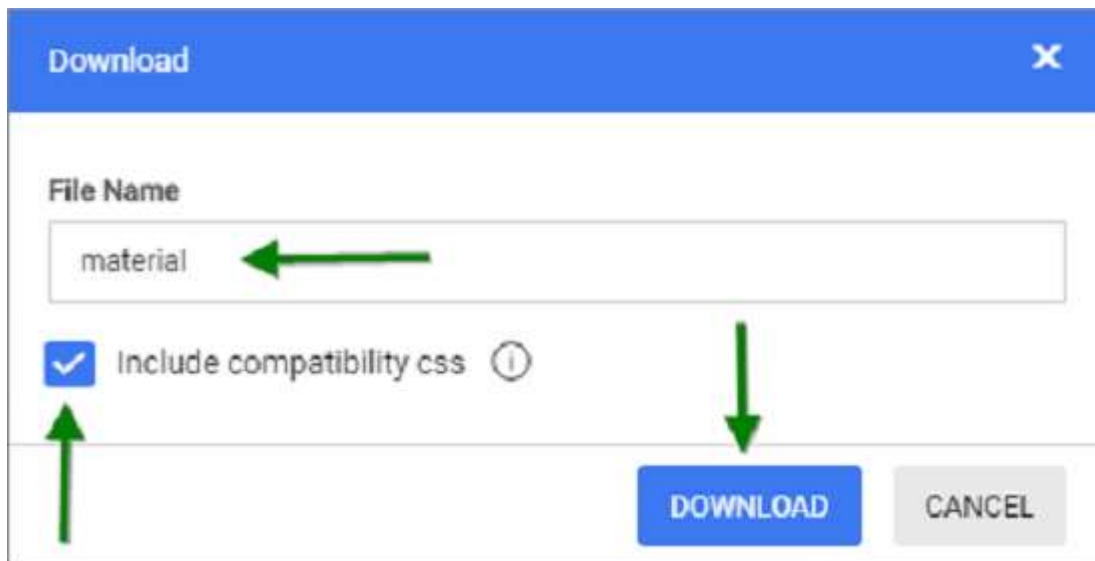
[Download the customized theme](#)

You can download the custom styles after customizing the theme colors.

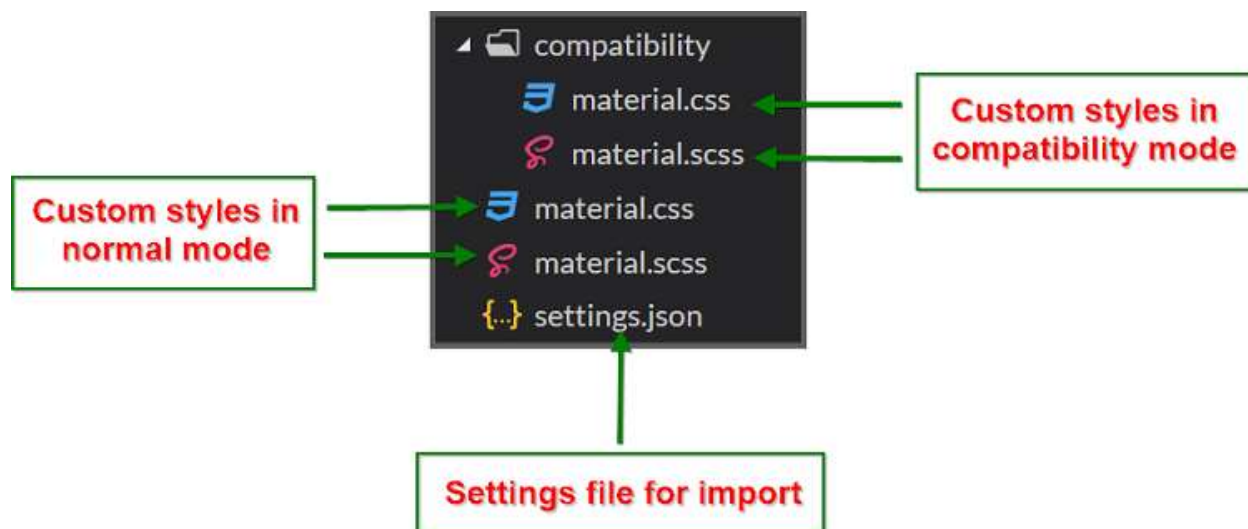
1. Click the Download button in the top right corner. The Download dialog will open.



2. Change the file name as you desire and then click download option.
 - **Include compatibility CSS** - If your application uses both Essential JS 1 and Essential JS 2 React components, then select the Include compatibility CSS check box before downloading the theme. This option will generate the custom theme for Essential JS 2 compatibility styles, which are compatible as Essential JS 1 styles.



Step 3: The download styles will come as a zip file that contains SCSS and CSS files for the selected Syncfusion React components. The current settings are stored in the `settings.json` file.



Using customized theme in a web application

You can directly use the customized CSS file in the web application.

1. Copy/paste the customized CSS file from the download folder into your application at any folder. Example: `styles/{file-name}.css`.
2. Refer the customized CSS file reference in the `index.html` or `shared/_layout.cshtml` main page head section.

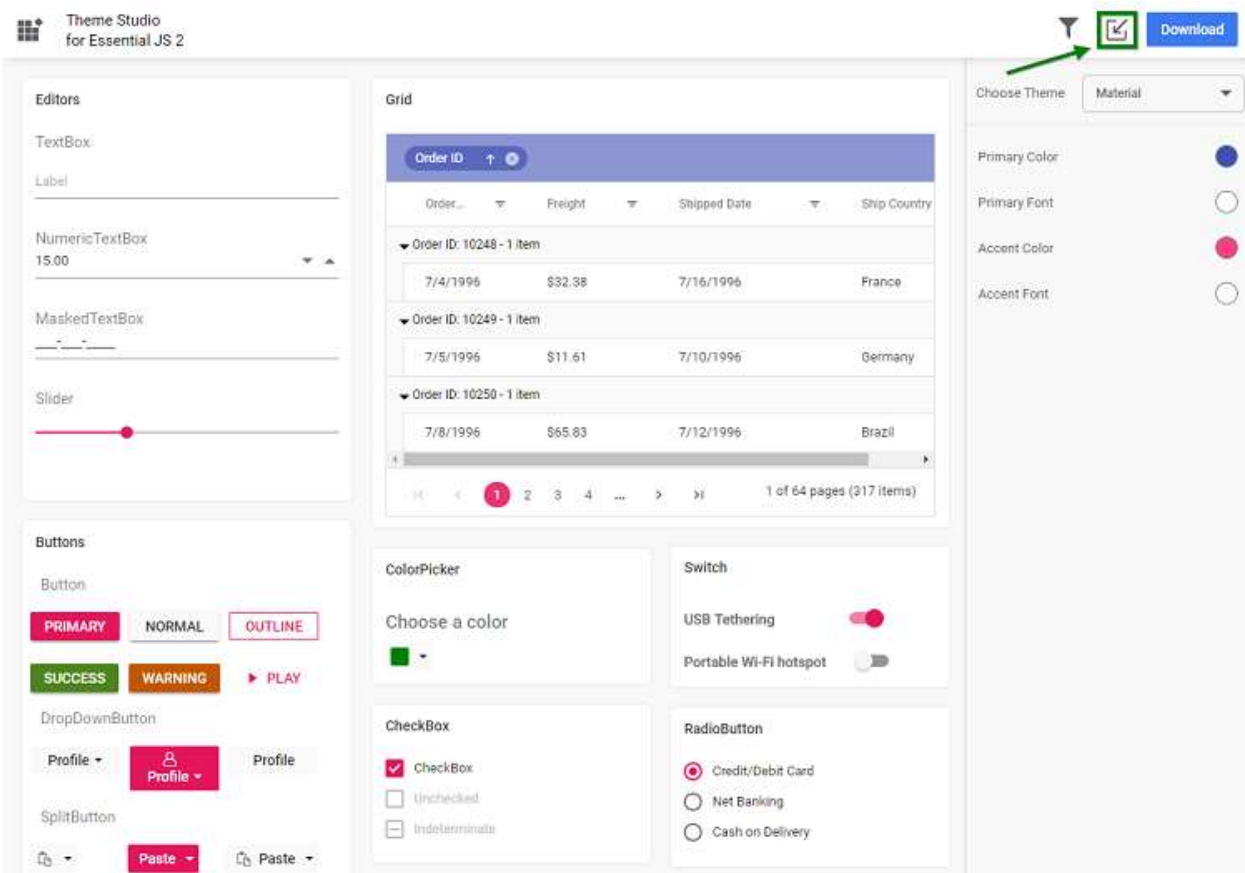
```
<head>
<link href="styles/{file-name}.css" rel="stylesheet"/>
</head>
```


If you are using Essential JS 1 and Essential JS 2 React components in a same web application, then you have to copy/paste the customized CSS file from the `compatibility` folder in the download location.

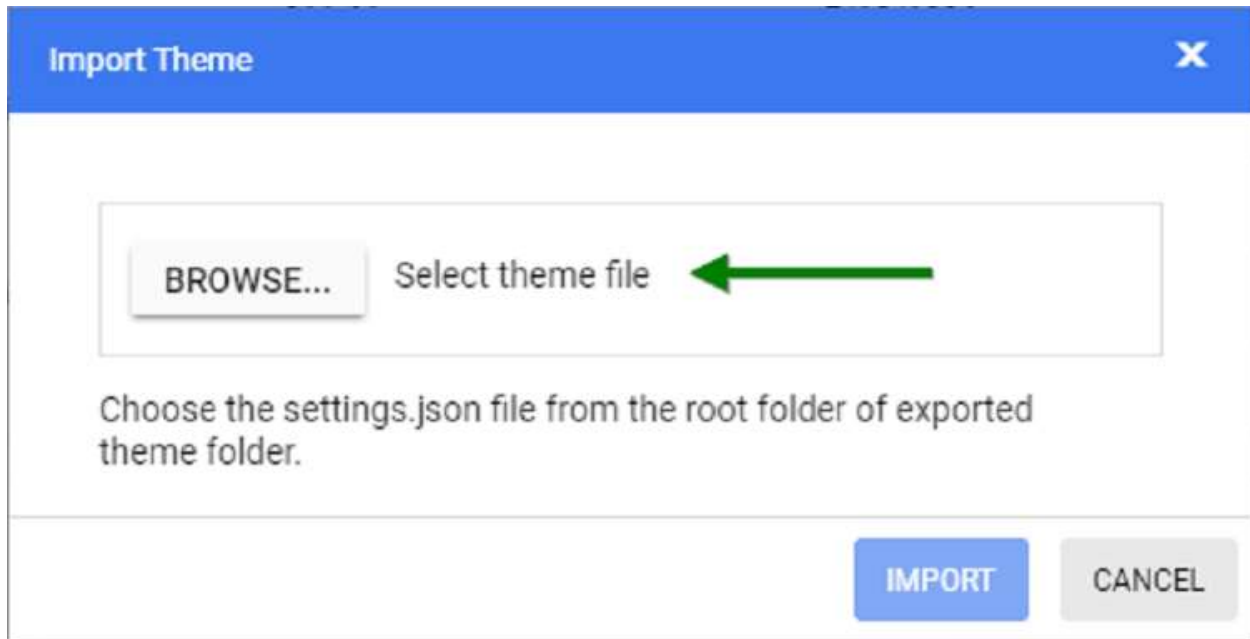
Import previously changed settings into the theme studio

When you want to change your application theme and UI design in the future, you won't need to customize the Syncfusion React components from scratch in the theme studio. Just import the old `settings.json` file to review and update your stored settings in the theme studio application.

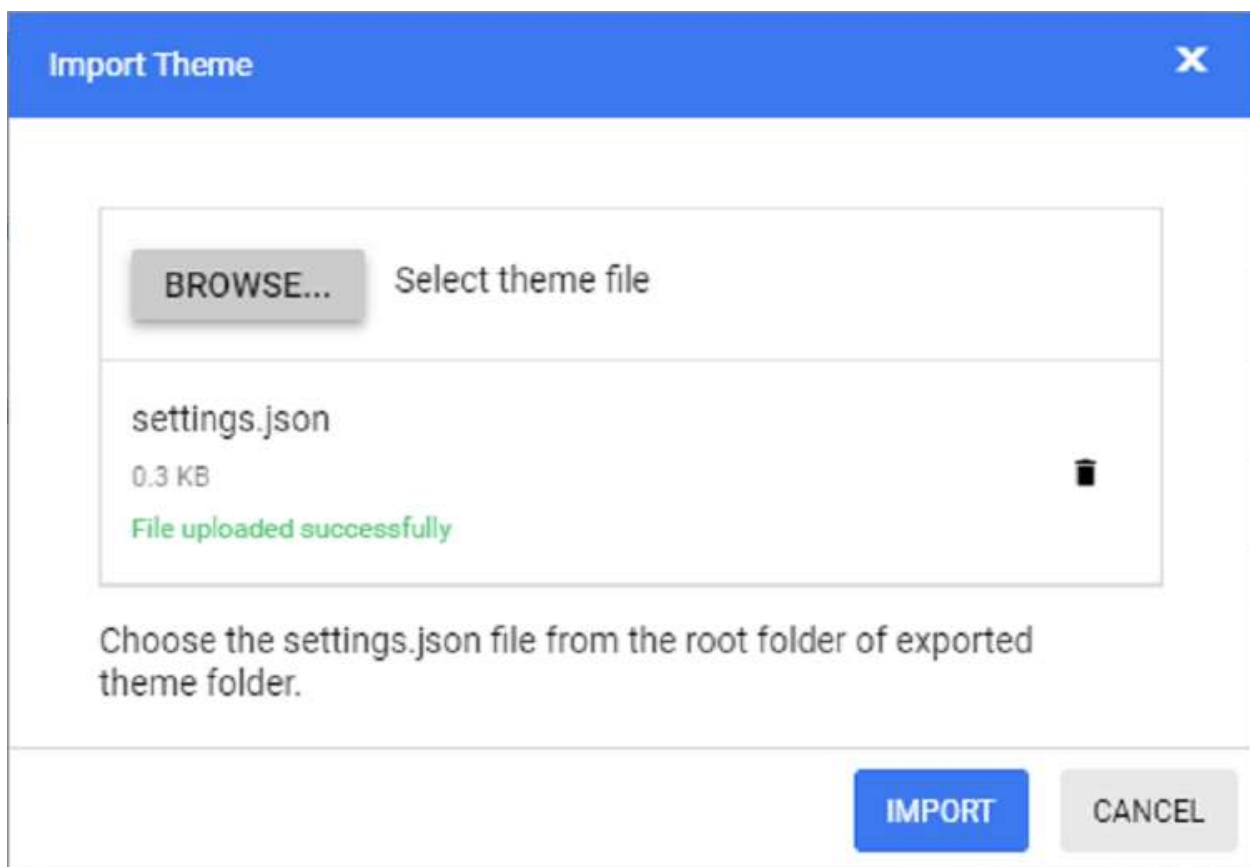
1. Click the Import icon in the top right corner.



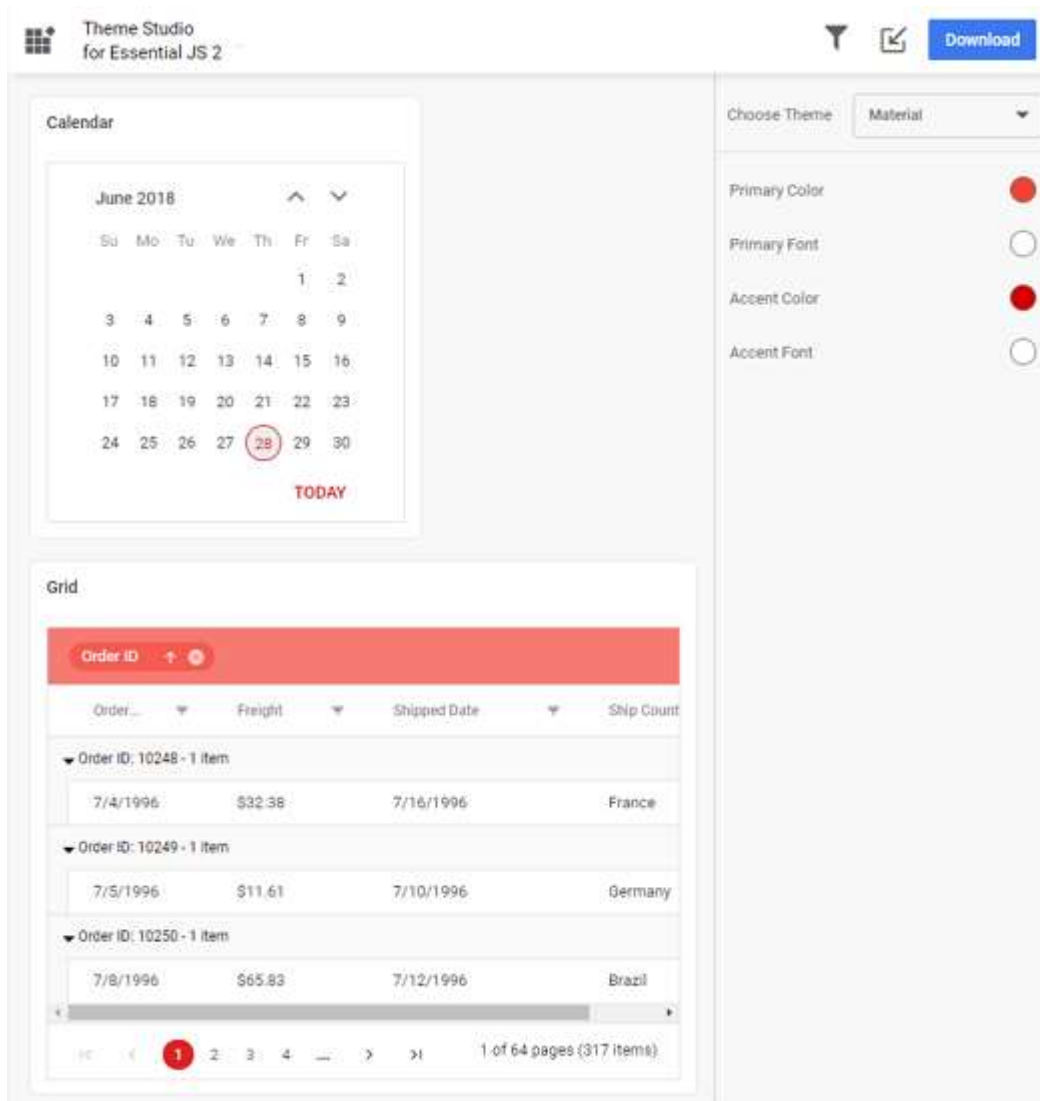
2. The Import Theme dialog will open. Click the Browse button and select a `settings.json` file you exported previously.



3. Click the Import button.



4. The stored data will be reflected in the theme studio application. Now you can change the theme colors based on your latest design and export the theme again.



- The exported file will contain your latest changes. You can just replace the older custom style with the newer one to refresh your application.

Common variables

The following list of common variables is used in the Syncfusion React library themes for all UI components. You can change these variables to customize the corresponding theme.

Bootstrap 5 theme

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000
\$white	#fff	#fff
\$gray-100	#f8f9fa	#f8f9fa

Name	Value (Default Theme)	Value (Dark Theme)
\$gray-200	#e9ecef	#e9ecef
\$gray-300	#dee2e6	#dee2e6
\$gray-400	#ced4da	#ced4da
\$gray-500	#adb5bd	#adb5bd
\$gray-600	#6c757d	#6c757d
\$gray-700	#495057	#495057
\$gray-800	#343a40	#343a40
\$gray-900	#212529	#212529
\$blue	#0d6efd	#0d6efd
\$indigo	#6610f2	#6610f2
\$purple	#6f42c1	#6f42c1
\$pink	#d63384	#d63384
\$red	#dc3545	#dc3545
\$orange	#fd7e14	#fd7e14
\$yellow	#ffc107	#ffc107
\$green	#198754	#198754
\$teal	#20c997	#20c997
\$cyan	#0dcaf0	#0dcaf0

Bootstrap 4 theme

Name	Value
\$white	#fff
\$gray-100	#f8f9fa

Name	Value
\$gray-200	#e9ecef
\$gray-300	#dee2e6
\$gray-400	#ced4da
\$gray-500	#adb5bd
\$gray-600	#6c757d
\$gray-700	#495057
\$gray-800	#343a40
\$gray-900	#212529
\$black	#000
\$blue	#007bff
\$indigo	#6610f2
\$purple	#6f42c1
\$pink	#e83e8c
\$red	#dc3545
\$orange	#fd7e14
\$yellow	#ffc107
\$green	#28a745
\$teal	#20c997
\$cyan	#17a2b8

Bootstrap theme

Design based on bootstrap 3 theme.

Name	Value (Default Theme)	Value (Dark Theme)
\$brand-primary	#317ab9	#0070f0
\$brand-primary-darken-10	#3071a9	darken(\$brand-primary, 10%)
\$brand-primary-darken-15	#2a6496	darken(\$brand-primary, 20%)
\$brand-primary-darken-25	#1f496e	darken(\$brand-primary, 30%)
\$brand-primary-darken-35	#142f46	darken(\$brand-primary, 40%)
\$brand-primary-font	#fff	#fff
\$gray-base	#000	#1a1a1a
\$gray-darker	#222	#131313
\$gray-dark	#333	#2a2a2a
\$gray	#555	#313131
\$gray-light	#777	#393939
\$grey-44	#444	#414141
\$grey-88	#888	#484848
\$grey-99	#999	#505050
\$grey-8c	#8c8c8c	#585858
\$grey-ad	#adadad	#676767
\$grey-dark-font	#fff	#f0f0f0
\$grey-white	#fff	#6e6e6e
\$grey-lighter	#eee	#767676
\$grey-f9	#f9f9f9	#7e7e7e
\$grey-f8	#f8f8f8	#858585
\$grey-f5	#f5f5f5	#8d8d8d

Name	Value (Default Theme)	Value (Dark Theme)
\$grey-e6	#e6e6e6	#959595
\$grey-dd	#ddd	#9c9c9c
\$grey-d4	#d4d4d4	#a4a4a4
\$grey-cc	#ccc	#acacac
\$grey-light-font	#333	#fff
\$brand-success	#5cb85c	#48b14c
\$brand-success-dark	#3c763d	#358238
\$brand-info	#5bc0de	#2aaac0
\$brand-info-dark	#31708f	#208090
\$brand-warning	#f0ad4e	#fac168
\$brand-warning-dark	#8a6d3b	#f9ad37
\$brand-danger	#d9534f	#d44f4f
\$brand-danger-dark	#a94442	#c12f2f
\$brand-success-light	#dff0d8	#dff0d8
\$brand-info-light	#d9edf7	#d9edf7
\$brand-warning-light	#fcf8e3	#fcf8e3
\$brand-danger-light	#f2dede	#f2dede
\$input-border-focus	#66afe9	#104888
\$brand-success-font	#3c763d	#2f7432
\$brand-info-font	#31708f	#1a6c7a
\$brand-warning-font	#8a6d3b	#9d6106
\$brand-danger-font	#a94442	#ac2a2a

Name	Value (Default Theme)	Value (Dark Theme)
\$base-font	#000	#000
\$brand-primary-lighten-10		lighten(\$brand-primary, 10%)
\$brand-primary-lighten-15		lighten(\$brand-primary, 15%)
\$brand-primary-lighten-20		lighten(\$brand-primary, 20%)
\$brand-primary-lighten-30		lighten(\$brand-primary, 30%)
\$brand-primary-lighten-40		lighten(\$brand-primary, 40%)

Material theme

Name	Value (Default Theme)	Value (Dark Theme)
\$accent	#e3165b	#ff80ab
\$accent-font	fff	#000
\$primary	#3f51b5	#3f51b5
\$primary-50	#e8eaf6	#e8eaf6
\$primary-100	#c5cae9	#c5cae9
\$primary-200	#9fa8da	#9fa8da
\$primary-300	#7986cb	#7986cb
\$primary-font	fff	fff
\$primary-50-font	#000	#000
\$primary-100-font	#000	#000
\$primary-200-font	#000	#000
\$primary-300-font	fff	fff
\$grey-white	fff	fff
\$grey-black	#000	#000

Name	Value (Default Theme)	Value (Dark Theme)
\$grey-50	#fafafa	#fafafa
\$grey-100	#f5f5f5	#f5f5f5
\$grey-200	#eee	#eee
\$grey-300	#e0e0e0	#e0e0e0
\$grey-400	#bdbdbd	#bdbdbd
\$grey-500	#9e9e9e	#9e9e9e
\$grey-600	#757575	#757575
\$grey-700	#616161	#616161
\$grey-800	#424242	#424242
\$grey-900	#212121	#212121
\$grey-dark	#303030	#303030
\$grey-light-font	#000	#000
\$grey-dark-font	#fff	#fff
\$base-font	#000	#000
\$error-font	#f44336	#ff6652
\$success-bg		#4caf50
\$error-bg		#ff6652
\$warning-bg		#ff9800
\$info-bg		#03a9f4
\$message-font		#fff
\$success-font		#4caf50
\$warning-font		#ff9800

Name	Value (Default Theme)	Value (Dark Theme)
\$info-font		#03a9f4

Tailwind CSS theme

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000
\$white	#fff	#fff
\$transparent	transparent	transparent
\$cool-gray-50	#f9fafb	#f9fafb
\$cool-gray-100	#f3f4f6	#f3f4f6
\$cool-gray-200	#e5e7eb	#e5e7eb
\$cool-gray-300	#d1d5db	#d1d5db
\$cool-gray-400	#9ca3af	#9ca3af
\$cool-gray-500	#6b7280	#6b7280
\$cool-gray-600	#4b5563	#4b5563
\$cool-gray-700	#374151	#374151
\$cool-gray-800	#1f2937	#1f2937
\$cool-gray-900	#111827	#111827
\$red-100	#fee2e2	#fee2e2
\$red-400	#f87171	#f87171
\$red-500	#ef4444	#ef4444
\$red-600	#dc2626	#dc2626
\$red-800	#991b1b	#991b1b
\$green-100	#dcfce7	#dcfce7

Name	Value (Default Theme)	Value (Dark Theme)
\$green-500	#22c55e	#22c55e
\$green-600	#16a34a	#16a34a
\$green-700	#15803d	#15803d
\$orange-100	#ffedd5	#ffedd5
\$orange-500	#f97316	#f97316
\$orange-600	#ea580c	#ea580c
\$orange-700	#c2410c	#c2410c
\$orange-800	#9a3412	#9a3412
\$cyan-300	#67e8f9	#67e8f9
\$cyan-400	#22d3ee	#22d3ee
\$cyan-500	#06b6d4	#06b6d4
\$cyan-600	#0891b2	#0891b2
\$cyan-800	#155e75	#155e75
\$indigo-50	#eef2ff	
\$indigo-100	#e0e7ff	
\$indigo-200	#c7d2fe	
\$indigo-300	#a5b4fc	
\$indigo-400	#818cf8	
\$indigo-500	#6366f1	
\$indigo-600	#4f46e5	
\$indigo-700	#4338ca	
\$indigo-800	#3730a3	

Name	Value (Default Theme)	Value (Dark Theme)
\$indigo-900	#312e81	
\$green-400		#4ade80
\$light-blue-50		#f0f9ff
\$light-blue-100		#e0f2fe
\$light-blue-400		#38bdf8
\$light-blue-500		#0ea5e9
\$light-blue-600		#0284c7
\$light-blue-700		#0369a1
\$light-blue-800		#075985

Microsoft Office Fabric theme

Name	Value (Default Theme)	Value (Dark Theme)
\$theme-primary	#0078d6	#0074cc
\$theme-dark-alt	darken(\$theme-primary, 3%)	darken(\$theme-primary, 3%)
\$theme-dark	darken(\$theme-primary, 10%)	darken(\$theme-primary, 6%)
\$theme-darker	darken(\$theme-primary, 18%)	darken(\$theme-primary, 10%)
\$theme-secondary	lighten(\$theme-primary, 3%)	lighten(\$theme-primary, 3%)
\$theme-tertiary	lighten(\$theme-primary, 21%)	lighten(\$theme-primary, 21%)
\$theme-light	lighten(\$theme-primary, 44%)	lighten(\$theme-primary, 44%)
\$theme-lighter	lighten(\$theme-primary, 49%)	lighten(\$theme-primary, 49%)
\$theme-lighter-alt	lighten(\$theme-primary, 55%)	lighten(\$theme-primary, 55%)
\$neutral-white	#fff	#201f1f
\$neutral-lighter-alt	#f8f8f8	#282727

Name	Value (Default Theme)	Value (Dark Theme)
\$neutral-lighter	#f4f4f4	#333232
\$neutral-light	#eaeaea	#414040
\$neutral-quintenaryalt	#dadada	#4a4848
\$neutral-quintenary	#d0d0d0	#514f4f
\$neutral-tertiary-alt	#c8c8c8	#6f6c6c
\$neutral-tertiary	#a6a6a6	#9a9a9a
\$neutral-secondary-alt	#767676	#c8c8c8
\$neutral-secondary	#666	#dadada
\$neutral-primary	#333	#fff
\$neutral-dark	#212121	#f4f4f4
\$neutral-black	#000	#f8f8f8
\$alert-bg	#deecf9	#bf7500
\$error-bg	#fde7e9	#cd2a19
\$success-bg	#dff6dd	#37844d
\$theme-dark-font	#fff	#fff
\$theme-primary-font	#fff	#fff
\$theme-light-font	#333	#000
\$neutral-light-font	#333	#dadada
\$neutral-light-fontalt	#000	#fff
\$grey-dark-font	#fff	#000
\$base-font	#333	#dadada
\$message-font	#333	#fff

Name	Value (Default Theme)	Value (Dark Theme)
\$alert-font	#d83b01	#ff9d48
\$error-font	#a80000	#ff5f5f
\$success-font	#107c10	#8eff8d
\$info-bg		#1e79cb
\$info-font		#62cfff

High Contrast theme

Name	Value
\$selection-bg	#ffd939
\$selection-font	#000
\$selection-border	#ffd939
\$hover-bg	#685708
\$hover-font	#fff
\$hover-border	#fff
\$border-default	#969696
\$border-alt	#757575
\$border-fg	#fff
\$border-fg-alt	#ffd939
\$bg-base-0	#000
\$bg-base-5	#0d0d0d
\$bg-base-10	#1a1a1a
\$bg-base-15	#262626
\$bg-base-20	#333

Name	Value
\$bg-base-75	#bfbfbf
\$bg-base-100	#fff
\$header-font	#ffd939
\$header-font-alt	#fff
\$content-font	#fff
\$content-font-alt	#969696
\$link	#8a8aff
\$invert-font	#000
\$success-bg	#166600
\$error-bg	#b30900
\$message-font	#fff
\$alert-bg	#944000
\$info-bg	#0056b3
\$success-alt	#2bc700
\$error-alt	#ff6161
\$alert-alt	#ff7d1a
\$info-alt	#66b0ff
\$disable	#757575

Fluent theme

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000
\$white	#fff	#fff

Name	Value (Default Theme)	Value (Dark Theme)
\$gray220	#11100f	#11100f
\$gray210	#161514	#161514
\$gray200	#1b1a19	#1b1a19
\$gray190	#201f1e	#201f1e
\$gray180	#252423	#252423
\$gray170	#292827	#292827
\$gray160	#323130	#323130
\$gray150	#3b3a39	#3b3a39
\$gray140	#484644	#484644
\$gray130	#605e5c	#605e5c
\$gray120	#797775	#797775
\$gray110	#8a8886	#8a8886
\$gray100	#979593	#979593
\$gray90	#a19f9d	#a19f9d
\$gray80	#b3b0ad	#b3b0ad
\$gray70	#bebbb8	#bebbb8
\$gray60	#c8c6c4	#c8c6c4
\$gray50	#d2d0ce	#d2d0ce
\$gray40	#e1dfdd	#e1dfdd
\$gray30	#edebe9	#edebe9
\$gray20	#f3f2f1	#f3f2f1
\$gray10	#faf9f8	#faf9f8

Name	Value (Default Theme)	Value (Dark Theme)
\$cyanblue10	#0078d4	#0078d4
\$red10	#d13438	#d13438
\$orange20	#ca5010	#ca5010
\$green20	#0b6a0b	#0b6a0b
\$cyan20	#038387	#038387

Styled-Component support

Syncfusion React components allow you to enhance the styling using [styled-component library](#).

Add styled component to the application

Import styled-components in the `src/App.tsx` file. To style our Syncfusion React component, pass the component in `styled` factory and override the EJ2 component styles. Here, `StyledButton` is the styled component.

```
`ts
const StyledButton = styled(ButtonComponent)`
&.e-btn {
background: #75e1ef;
color: #000000;
}
`;
```

Refer to the below code to create a styled button.

```
`ts
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import styled from 'styled-components';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
function App() {
const StyledButton = styled(ButtonComponent)`
&.e-btn {
background: #75e1ef;
color: #000000;
```

```

}
`;
render() {
return (<div className='control-pane'>
<StyledButton>Button</StyledButton>
</div>);
}
}
export default App;
ReactDOM.render(<App />,document.getElementById('sample'));
`

```

Dynamically computed props styling

We can style the Syncfusion React components dynamically based on props.

```

`ts
const StyledButton = styled(ButtonComponent)`
&.e-btn {
${props => props.disabled && css`
background: palevioletred;
color: white;
`}
}
`;
`

```

The below example code shows the dynamically styling the components using props.

```

`ts
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import styled from 'styled-components';
import { css } from 'styled-components'
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
function App() {
const StyledButton = styled(ButtonComponent)`
&.e-btn {

```

```

${props => props.disabled && css`
background: palevioletred;
color: white;
`}
}
`;
return (<div className='control-pane'>
<StyledButton disabled={true}>Button</StyledButton>
</div>);
}
export default App;
ReactDOM.render(<App />,document.getElementById('sample'));
`

```

Material 3 Theme with CSS Variables

[Material 3](#) is the latest version of Google's open-source design system, Material Design. It's the successor to [Material 2](#), which is a popular design system used in many apps. It has been specifically designed to align seamlessly with the new visual style and system UI introduced in Android 12 and above.

[CSS variables](#), also known as custom properties, are entities defined by CSS authors that contain specific values that can be reused throughout a CSS file. They are identified by their name, which must begin with two hyphens (--) followed by an identifier. These variables can be assigned any valid CSS value, such as colors, lengths, or fonts. To retrieve the value of a CSS variable, the `var()` function is used.

Material 3 - Syncfusion React Components

Syncfusion has introduced the Material 3 theme across all EJ2 Components, featuring both **light** and **dark** variants. This theme utilizes **CSS variables** to allow easy customization of Component colors in CSS format. With this implementation, users can seamlessly switch between light and dark color schemes, providing a flexible solution to meet their preferences and application needs.

Kindly note that in the Material 3 theme, Syncfusion utilizes CSS variables with `rgb()` values for color variables. The use of hex values in this context may lead to improper functionality. For example, in previous versions of the Material theme or other themes, the primary color variable was defined as follows: `$primary: #6200ee;`. However, in the Material 3 theme, the primary color variable is defined as follows: `--color-sf-primary: 98, 0, 238;`.

How does Syncfusion Material 3 theme utilize CSS Variables?

Syncfusion's Material 3 theme incorporates support for CSS variables, utilizing `rgb()` values for customizing colors. For more information you can refer this [documentation](#) for color variables of material 3 theme.

```

`CSS
:root {

```

```

--color-sf-black: 0, 0, 0;
--color-sf-white: 255, 255, 255;
--color-sf-primary: 103, 80, 164;
--color-sf-primary-container: 234, 221, 255;
--color-sf-secondary: 98, 91, 113;
--color-sf-secondary-container: 232, 222, 248;
--color-sf-tertiary: 125, 82, 96;
--color-sf-tertiary-container: 255, 216, 228;
--color-sf-surface: 255, 255, 255;
--color-sf-surface-variant: 231, 224, 236;
--color-sf-background: var(--color-sf-surface);
--color-sf-on-primary: 255, 255, 255;
--color-sf-on-primary-container: 33, 0, 94;
--color-sf-on-secondary: 255, 255, 255;
--color-sf-on-secondary-container: 30, 25, 43;
--color-sf-on-tertiary: 255, 255, 255;
}
`

```

How to get the Material 3 theme?

To access the Material 3 theme provided by Syncfusion, you have two primary options,

- Package
- CDN links

```

| | Light | Dark |
|-----|-----|-----|
|Package | Material 3 Light | Material 3 Dark |
| CDN   | Material 3 Light | Material 3 Dark |

```

Color Customization in Material 3

CSS variables allows you to dynamically change color values in real-time using JavaScript. This flexibility enables you to create interactive experiences where colors can adjust based on user interactions or other dynamic factors.

Customization using CSS

Here you can find the example for Material 3 customization using Css class.

INDEX.JSX

```

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

```

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
function App() {
    return (<div>
        {/* Primary Button - Used to represent a primary action. */}
        <ButtonComponent cssClass='e-primary'>button</ButtonComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('button'));
```

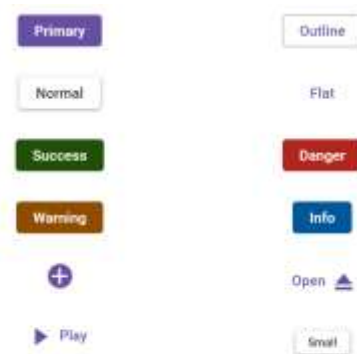
INDEX.TSX

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
enableRipple(true);
function App() {
    return (
        <div>
            { /* Primary Button - Used to represent a primary action. */ }
            <ButtonComponent cssClass='e-primary'>Button</ButtonComponent>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('button'));
```

INDEX.CSS

```
:root {
    --color-sf-primary: 104, 134, 164;
}
button {
    margin: 25px 5px 20px 20px;
}
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
```

Default primary value



```

:root {
  --color-sf-black: 0, 0, 0;
  --color-sf-white: 255, 255, 255;
  --color-sf-primary: 255, 255, 255;
  --color-sf-primary-container: 234, 221, 255;
  --color-sf-secondary: 98, 81, 113;
  --color-sf-secondary-container: 232, 222, 248;
  --color-sf-tertiary: 125, 82, 96;
  --color-sf-tertiary-container: 255, 216, 228;
  --color-sf-surface: 279, 255, 255;
  --color-sf-surface-variant: 231, 224, 236;
  --color-sf-background: var(--color-sf-surface);
  --color-sf-on-primary: 255, 255, 255;
  --color-sf-on-primary-container: 33, 0, 94;
  --color-sf-on-secondary: 255, 255, 255;
  --color-sf-on-secondary-container: 30, 25, 43;
  --color-sf-on-tertiary: 255, 255, 255;
  --color-sf-on-tertiary-container: 55, 11, 30;
  --color-sf-on-surface: 28, 27, 31;
  --color-sf-on-surface-variant: 73, 69, 78;
  --color-sf-on-background: 38, 27, 31;
  --color-sf-outline: 123, 110, 128;
  --color-sf-outline-variant: 106, 106, 107;
  --color-sf-shadow: 0, 0, 0;
  --color-sf-surface-tint-color: 100, 0, 100;
  --color-sf-inverse-surface: 48, 48, 51;
}

```

Customized primary value



```

:root {
  --color-sf-black: 0, 0, 0;
  --color-sf-white: 255, 255, 255;
  --color-sf-primary: 255, 255, 255;
  --color-sf-primary-container: 234, 221, 255;
  --color-sf-secondary: 98, 81, 113;
  --color-sf-secondary-container: 232, 222, 248;
  --color-sf-tertiary: 125, 82, 96;
  --color-sf-tertiary-container: 255, 216, 228;
  --color-sf-surface: 279, 255, 255;
  --color-sf-surface-variant: 231, 224, 236;
  --color-sf-background: var(--color-sf-surface);
  --color-sf-on-primary: 255, 255, 255;
  --color-sf-on-primary-container: 33, 0, 94;
  --color-sf-on-secondary: 255, 255, 255;
  --color-sf-on-secondary-container: 30, 25, 43;
  --color-sf-on-tertiary: 255, 255, 255;
  --color-sf-on-tertiary-container: 55, 11, 30;
  --color-sf-on-surface: 28, 27, 31;
  --color-sf-on-surface-variant: 73, 69, 78;
  --color-sf-on-background: 38, 27, 31;
  --color-sf-outline: 123, 110, 128;
  --color-sf-outline-variant: 106, 106, 107;
  --color-sf-shadow: 0, 0, 0;
  --color-sf-surface-tint-color: 100, 0, 100;
  --color-sf-inverse-surface: 48, 48, 51;
}

```

With this CSS variable support, you can effortlessly customize the color variable values for Syncfusion React Components.

Material 3 Light Theme

Syncfusion has implemented the Material 3 theme, offering both Light and Dark variants. In the Material 3 light theme, there are distinct class variables for both light and dark modes, providing flexibility for seamless switching between the two modes within your application.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useState } from 'react';
import { ButtonComponent, CheckBoxComponent } from '@syncfusion/ej2-react-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
function App() {
  const [isChecked, setIsChecked] = useState(false);
  const handleCheckboxChange = (event) => {
    setIsChecked(event.target.checked);
    // Add or remove the classes on the body element based on the checkbox state
    const body = document.body;
    if (event.target.checked) {
      body.classList.add('dark');
      body.classList.add('dark');
    } else {

```

```

        body.classList.remove('e-dark-mode');
        body.classList.remove('e-dark-mode');
    }
};
return (<div>
    /* Primary Button - Used to represent a primary action. */
    <CheckBoxComponent label="Enable Darkmode" checked={isChecked}
onChange={handleCheckboxChange}/><br/>
    <ButtonComponent cssClass='e-primary'>button</ButtonComponent>
    /* Success Button - Used to represent a positive action. */
    <ButtonComponent cssClass='e-success'>button</ButtonComponent>
    /* Info Button - Used to represent an informative action. */
    <ButtonComponent cssClass='e-info'>button</ButtonComponent>
    /* Warning Button - Used to represent an action with caution.*/
    <ButtonComponent cssClass='e-warning'>button</ButtonComponent>
    /* Danger Button - Used to represent a negative action.*/
    <ButtonComponent cssClass='e-danger'>button</ButtonComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('button'));

```

INDEX.TSX

```

import {ButtonComponent, CheckBoxComponent} from '@syncfusion/ej2-react-
buttons';
import { enableRipple } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useState } from 'react';
enableRipple(true);
function App() {
    const [isChecked, setIsChecked] = useState(false);
    const handleCheckboxChange = (event: React.ChangeEvent<HTMLInputElement>)
=> {
        setIsChecked(event.target.checked);
        // Add or remove the class on the body element based on the checkbox
state
        if (event.target.checked) {
            document.body.classList.add('dark');
            document.body.classList.add('e-dark-mode');
        } else {
            document.body.classList.remove('dark');
            document.body.classList.remove('e-dark-mode');
        }
    };
    return (
        <div>
            /* checkbox - Used to represent checkbox. */
            <CheckBoxComponent label="Enable Darkmode" checked={isChecked}
onChange={handleCheckboxChange}/><br/>
            /* Primary Button - Used to represent a primary action. */
            <ButtonComponent cssClass='e-primary'>Button</ButtonComponent>
            /* Success Button - Used to represent a positive action. */
            <ButtonComponent cssClass='e-success'>Button</ButtonComponent>
            /* Info Button - Used to represent an informative action. */

```

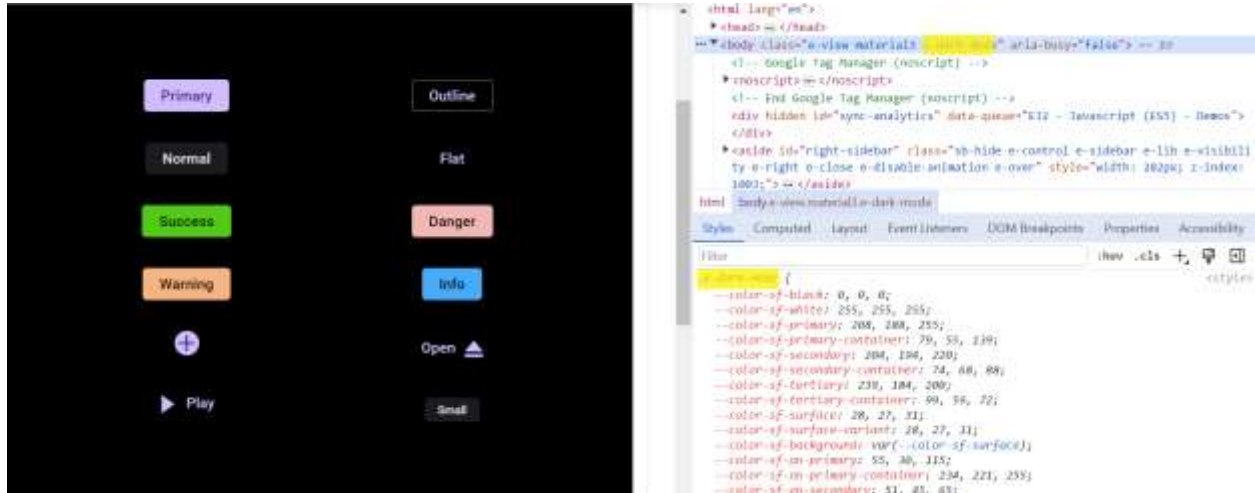
```

        <ButtonComponent cssClass='e-info'>Button</ButtonComponent>
        { /* Warning Button - Used to represent an action with caution.*/ }
        <ButtonComponent cssClass='e-warning'>Button</ButtonComponent>
        { /* Danger Button - Used to represent a negative action.*/ }
        <ButtonComponent cssClass='e-danger'>Button</ButtonComponent>
    </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('button'));

```

How to switch dark mode?

With CSS variable support, transitioning between light and dark theme modes has become effortless. To activate dark mode, just append the `e-dark-mode` class to the body section of your application. Once applied, the theme seamlessly switches to dark mode. Please refer to the example image below for visual guidance.



ThemeStudio Application

The ThemeStudio application now includes seamless integration with the Material 3 theme, offering a comprehensive solution for customization requirements. This enhancement enables users to effortlessly customize and personalize their themes.

Access the Syncfusion ThemeStudio application, featuring the Material 3 theme, via the following link:

[Link to Syncfusion ThemeStudio with Material 3 Theme](#)

Common

Rendering Engine of Syncfusion React Components

The Syncfusion React component acts as a wrapper component that renders UI based on a pure JavaScript engine. All component functionalities are developed using the JavaScript framework, resulting in easier, faster, and more efficient development.

When using the Syncfusion React component, only the root element of the component is rendered through the React virtual DOM, while other DOM elements are rendered through JavaScript. This approach is explained in detail in the wrapper component documentation, which you can access at [this link](#).

Changes made to the DOM outside of React are not recognized by React. To avoid conflicts, Syncfusion utilizes React component lifecycle methods to ensure that all functionalities of the component work properly within the React framework. And our React components support all React framework-related features.

Individual Components Script Dependency

When utilizing **Webpack externals**, it is important to specify the right script reference order as well as its dependencies. Syncfusion React components each have their own set of dependencies, which may include other Syncfusion packages.

There are two ways to refer to scripts: through **CDN** links or **NPM** packages. Refer to the table below to discover the correct CDN script reference sequence and its dependencies for a certain Syncfusion React component. If using **NPM** packages, replace the **CDN** link with the local path of the relevant scripts.

Components	Scripts
Accordion , App Bar , Bread crumb , Carousel , Context Menu , Menu , Sidebar , Tab , Toolbar , TreeView	
Accumulation Chart , Smith Chart , Chart , Sparkline , Range Navigator , Stock Chart , Bullet Chart	
Auto Complete , Combo Box , Dropdown List , Dropdown Tree , ListBox , Mention , Multiselect	
Barcode	
Calendar , Date Picker , DateRangePicker List , DateTimePicker , TimePicker	

Check Box , Chips , FloatingActionButton , Radio Button , Speed Dial , Switch	
Circular Gauge	
Color Picker , MaskedTextBox , NumericTextBox , Range Slider , Rating , Text Box , Uploader	
Dashboard Layout , Splitter	
Dialog , Predefined Dialogs , Tooltip	
Document Editor	
DropDownButton , ProgressButton , SplitButton	
File Manager	
Gantt	
Grid	
HeatMap Chart	
Image Editor	
InplaceEditor	
Kanban	
Linear Gauge	
ListView	
Maps	
Message , Skeleton , Toast	

PivotTable	
ProgressBar	
Query Builder	
Ribbon	
RichTextEditor	
Schedule	
Spreadsheet	
TreeMap	

Accessibility in Syncfusion React Components

Accessibility overview

Accessibility in components refers to the practice of designing and building user interface elements in a way that makes them accessible to users with disabilities. This can include a variety of things, such as making sure that text is high-contrast and easy to read, providing alternative text for images, and designing controls and interactions that can be used with a keyboard or assistive technology.

Accessibility standards

The component is said to be accessible when it is constructed in accordance with certain standards that are required to make it accessible.

The accessibility of the components consists of the following standards and aspects:

- [ADA](#) - A law to ensure that people with disabilities have the same opportunities and access as people without disabilities.
- [WCAG 2.2](#) - The Web Content Accessibility Guidelines (WCAG) provide guidelines developed by the World Wide Web Consortium (W3C) to ensure web content is accessible to people with disabilities. [WCAG 2.2](#) establishes a framework of accessibility principles and their associated success criteria. The level of accessibility conformance achieved by a web application is determined by the extent to which it meets these success criteria, categorized into three levels: A, AA, and AAA.
- [Section 508](#) - It is a set of guidelines for making electronic and information technology (EIT) accessible to people with disabilities. These standards apply to federal agencies in the United States, and they are based on the Web Content Accessibility Guidelines (WCAG).
- [WAI-ARIA](#) - WAI-ARIA stands for "Web Accessibility Initiative - Accessible Rich Internet Applications." It is a set of technical specifications and guidelines developed by the World Wide Web Consortium (W3C) as part of the Web Accessibility Initiative (WAI). WAI-ARIA is designed to enhance the accessibility of dynamic web content, particularly web applications and rich internet applications (RIAs), for people with disabilities. WAI-ARIA provides a set of roles, states, and properties that can be added to HTML elements to provide additional context and information about the purpose and behavior of those elements. This can help assistive technologies better understand and interpret web content and interact with web applications.

- [Keyboard navigation](#) - It refers to the ability to use a keyboard to interact with and navigate through a user interface. It is an important aspect of web accessibility, as it allows people who cannot use a mouse or other pointing device to access and use web content and applications.

Syncfusion React components adhere to these established standards.

Accessibility compliance

The accessibility support provided by Syncfusion React components is based on a collection of methodologies for adhering to and applying [recognized standards and technical specifications](#) to ensure an intuitive experience for people with disabilities.

There are several methodologies of accessibility validation that can be performed on the Syncfusion React components, such as:

- The [WAI-ARIA patterns](#) are followed by the Syncfusion React components to enable appreciable accessibility.
- Each React component is subjected to manual testing with a screen reader and also automated test cases to ensure the component's required attributes.
- Attributes are allocated and updated correctly during interaction as well. Each component has been assigned a distinct `Role` attribute and its own set of ARIA attributes defined by the [WCAG 2.2](#) specification.

In addition to the methodologies mentioned above, Syncfusion React components are constructed to support the following accessibility aspects.

Screen reader support

A screen reader allows people who are blind or visually impaired to use a computer by reading aloud the text that is displayed on the screen. Syncfusion React components followed the [WAI-ARIA](#) standards to work properly in the screen readers such as [Narrator](#) for Windows and [Embedded VoiceOver](#) for MAC.

Right-To-Left support

Right-to-Left (RTL) support allows applications to effectively communicate with users who use languages that are written from right to left, such as Arabic, Hebrew, etc. Syncfusion React components support the Right-to-Left feature. Refer to the [Right-to-Left documentation](#) to learn more about this support.

Color contrast

Syncfusion React components come equipped with [predefined themes](#) that guarantee the presence of satisfactory color contrast, benefiting individuals with visual impairments.

Mobile device support

Syncfusion React components are more user-friendly and accessible to individuals using mobile devices, including those with disabilities. These are designed to be responsive, adaptable to various screen sizes and orientations, and touch-friendly.

Keyboard navigation support

Syncfusion React components support keyboard navigation, allowing users who rely on alternate methods to effortlessly navigate and interact with the component.

Ensuring accessibility

Ensuring the accessibility of Syncfusion React components is crucial for making the product inclusive and user-friendly for individuals with disabilities. This process involves various types of accessibility testing, including:

- **Automated testing:** The Syncfusion React component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools.
- **Manual testing:** This type of testing involves manually evaluating the Syncfusion React components. During manual accessibility testing, testers will ensure accessibility using the screen readers such as [Narrator](#) for Windows and [Embedded VoiceOver](#) for MAC.

Syncfusion React components will keep improving when there is anything required. It also involves client feedback to make the component more accessible.

[Accessibility support for specific components](#)

Consult the component-specific documents below for detailed information about how Syncfusion React components ensure compliance with accessibility standards, encompassing Section 508, WAI-ARIA, WCAG 2.2, keyboard navigation, and more.

<style>

[table](#)

{

border:0 !important;

line-height: 2!important;

}

tr

{

border:0 !important;

}

td

{

border:0 !important;

vertical-align: top;

}

.controlanchorlink

{

text-decoration: none !important;

font-size: 14px !important;

text-align: left !important;

padding: 5px 0px;

letter-spacing: 1px;

}

.controlcategory

```

{
font-size: 14px !important;
text-align: left !important;
font-weight: bold !important;
letter-spacing: 0.7px;
}
}
</style>

```

| | | | |
|----------------------------------|------------------------------------|--------------------------------------|--------------------------------------|
| | DATA
VISUALIZATION | CALENDARS | DROPDOWNS |
| GRIDS | Accumulation Chart | Scheduler | AutoComplete |
| DataGrid | Charts | Calendar | ComboBox |
| Pivot Table | 3D Chart | DatePicker | DropDown List |
| Tree Grid | Stock Chart | DateRangePicker | DropDown Tree |
| Spreadsheet | Circular Gauge | DateTime Picker | Multiselect DropDown |
| FILE VIEWERS &
EDITORS | Diagram | TimePicker | Mention |
| In-place Editor | HeatMap Chart | Gantt Chart | ListBox |
| PDF Viewer | Linear Gauge | INPUTS | NAVIGATION |
| RichTextEditor | Maps | TextBox | Accordion |
| Word Processor | Range Selector | Input Mask | AppBar |
| Image Editor | Smith Chart | Masked TextBox | Breadcrumb |
| LAYOUT | Sparkline Charts | Numeric TextBox | Carousel |
| Dialog | TreeMap | Radio Button | Context Menu |
| ListView | Bullet Chart | CheckBox | Menu Bar |
| Tooltip | Kanban | Color Picker | Sidebar |
| Splitter | BUTTONS | File Upload | Tabs |
| Dashboard Layout | Button | Range Slider | Toolbar |
| | Button Group | Toggle Switch Button | Ribbon |
| | Dropdown Menu | Signature | TreeView |
| | Progress Button | Rating | File Manager |
| | Split Button | FORMS | Pager |
| | Chips | Query Builder | Stepper |

| | | | |
|--|----------------------------|--|------------------------------|
| | FAB | | NOTIFICATION |
| | Speed Dial | | Message |
| | | | Toast |
| | | | Progress Bar |
| | | | Skeleton |

Right-To-Left support in Syncfusion React Components

Right-to-Left (RTL) support allows applications to effectively communicate with users who use languages that are written from right to left, such as Arabic, Hebrew, etc.

Syncfusion React UI components support for right-to-left (RTL) by setting the `enableRtl` property to `true`. This adds the class name `e-rtl` to the component element and renders all Syncfusion React components in a right-to-left direction.

Enable RTL for all components

To enable Right-To-Left (RTL) support for all components, users can set the `enableRtl` property directly in their application. Here is an example code snippet using the ListView component:

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ListViewComponent } from '@syncfusion/ej2-react-lists';
import { enableRtl } from '@syncfusion/ej2-base';
// Enables Right to left alignment for all components
enableRtl(true);
function App() {
    const data = ["Artwork", "Abstract", "Modern Painting", "Ceramics",
    "Animation Art", "Oil Painting"];
    return (
        // specifies the tag to render the ListView component
        <ListViewComponent id='list' dataSource={data} showHeader='true'
        headerTitle='Painting types'></ListViewComponent>);
    }
    export default App;
    ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ListViewComponent } from '@syncfusion/ej2-react-lists';
import { enableRtl } from '@syncfusion/ej2-base';
// Enables Right to left alignment for all components
enableRtl(true);
function App() {
    const data = ["Artwork", "Abstract", "Modern Painting", "Ceramics",
    "Animation Art", "Oil Painting"];
    return (
```

```
// specifies the tag to render the ListView component
<ListViewComponent id='list' dataSource={data} showHeader = 'true'
headerTitle = 'Painting types' ></ListViewComponent>
);
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React ListView</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-
lists/styles/material.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>
</head>
<body>
  <div id='element'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

Enable RTL for an individual component

To enable Right-To-Left (RTL) support for an individual component, users can set the `enableRtl` property in the component's options. Here is an example code snippet using the `ListView` component:

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { ListViewComponent } from '@syncfusion/ej2-react-lists';
function App() {
  const data = ["Artwork", "Abstract", "Modern Painting", "Ceramics",
"Animation Art", "Oil Painting"];
  return (
    // specifies the tag to render the ListView component
    <ListViewComponent id='list' dataSource={data} showHeader='true'
headerTitle='Painting types' enableRtl='true'></ListViewComponent>);
  }
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```


INDEX.TSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { ListViewComponent } from '@syncfusion/ej2-react-lists';
function App() {
  const data = ["Artwork", "Abstract", "Modern Painting", "Ceramics",
    "Animation Art", "Oil Painting"];
  return (
    // specifies the tag to render the ListView component
    <ListViewComponent id='list' dataSource={data} showHeader = 'true'
    headerTitle = 'Painting types' enableRtl = 'true' ></ListViewComponent>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React ListView</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-
lists/styles/material.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>
</head>
<body>
  <div id='element'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

Animation in React

The **Animation** is used to perform animation effects on HTML elements by running a sequence of frames. It can be used to enhance the user experience.

Syncfusion [Animation](#) library supports animating the HTML elements using the [animate](#) method. This method adds the `e-animate`, `e-animation-id` attributes, and CSS style to the HTML element and removes them after the animation effect is completed.

Animation effects

An animation effect refers to the visual change that occurs over a period of time in HTML elements. The [Animation](#) library supports different types of animation [effects](#). The [name](#) property is used to specify the animation effect of an animation.

Here is an example code snippet using the `FadeOut` and `ZoomOut` animation effects:

INDEX.JSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect, useRef } from 'react';
import { Animation } from '@syncfusion/ej2-base';
function App() {
  let element1 = useRef();
  let element2 = useRef();
  useEffect(() => {
    let animation = new Animation();
    animation.animate(element1, { name: 'FadeOut' });
    animation.animate(element2, { name: 'ZoomOut' });
  }, []);
  return (
    <div id="container">
      <div id="element1" ref={ele => { element1 = ele; }}></div>
      <div id="element2" ref={ele => { element2 = ele; }}></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 { useEffect, useRef } from 'react';
import { Animation } from '@syncfusion/ej2-base';
function App() {
  let element1 = useRef();
  let element2 = useRef();
  useEffect(() => {
    let animation: Animation = new Animation();
    animation.animate(element1, { name: 'FadeOut' });
    animation.animate(element2, { name: 'ZoomOut' });
  }, [])
  return (
    <div id="container">
      <div id="element1" ref={ele => { element1 = ele; }}></div>
      <div id="element2" ref={ele => { element2 = ele; }}></div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

```
{% endraw %}
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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="index.css" rel="stylesheet" />
  <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://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>
</head>
<body>
  <div id='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

Animation duration

Animation [duration](#) is the animation property that specifies the length of time that an animation should take to complete. The animation duration is specified in milliseconds (ms) and determines the total amount of time that an animation should run.

For example, if an animation has a duration of 2 seconds, it will take 2 seconds to complete from start to finish. The duration of an animation affects the overall pace of the animation and can be adjusted to match the desired speed and style of the animation.

The value of the animation duration can be adjusted to change the speed of the animation, with shorter durations resulting in faster animations and longer durations resulting in slower animations.

Here is an example code snippet using the animation effects with a duration of **5000** milliseconds:

INDEX.JSX

```
{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect, useRef } from 'react';
import { Animation } from '@syncfusion/ej2-base';
function App() {
  let element1 = useRef();
```

```

    let element2 = useRef();
    useEffect(() => {
      let animation = new Animation({ duration: 5000 });
      animation.animate(element1, { name: 'FadeOut' });
      animation.animate(element2, { name: 'ZoomOut' });
    }, []);
    return (<div id="container">
      <div id="element1" ref={ele} => { element1 = ele; }></div>
      <div id="element2" ref={ele} => { element2 = ele; }></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 { useEffect, useRef } from 'react';
import { useRef } from 'react';
import { Animation } from '@syncfusion/ej2-base';
function App() {
  let element1 = useRef();
  let element2 = useRef();
  useEffect(() => {
    let animation: Animation = new Animation({ duration: 5000 });
    animation.animate(element1, { name: 'FadeOut' });
    animation.animate(element2, { name: 'ZoomOut' });
  }, [])
  return (
    <div id="container">
      <div id="element1" ref={ele} => { element1 = ele; }></div>
      <div id="element2" ref={ele} => { element2 = ele; }></div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% endraw %}

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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="index.css" rel="stylesheet" />
  <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://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>
</head>
<body>
  <div id='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Animation delay

The animation [delay](#) is the animation property that specifies the amount of time to wait before starting an animation. The animation delay is specified in milliseconds (ms) and determines the amount of time that should elapse before an animation begins.

For example, if an animation has a delay of 2 seconds, it will wait for 2 seconds before starting. This can be useful in creating more complex animations, where multiple elements are animated in sequence, or in creating animations that start only after a user interaction has taken place.

Here is an example code snippet using the animation effects with a delay of **2000** milliseconds:

INDEX.JSX

```

{% raw %}
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect, useRef } from 'react';
import { Animation } from '@syncfusion/ej2-base';
function App() {
  let element1 = useRef();
  let element2 = useRef();
  useEffect(() => {
    let animation = new Animation({ delay: 2000, duration: 5000 });
    animation.animate(element1, { name: 'FadeOut' });
    animation.animate(element2, { name: 'ZoomOut' });
  }, []);
  return (
    <div id="container">
      <div id="element1" ref={ele} => { element1 = ele; }></div>
      <div id="element2" ref={ele} => { element2 = ele; }></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 { useEffect, useRef } from 'react';
import { useRef } from 'react';
import { Animation } from '@syncfusion/ej2-base';
function App() {
  let element1 = useRef();
  let element2 = useRef();
  useEffect(() => {
    let animation: Animation = new Animation({ delay: 2000, duration: 5000
  });
  animation.animate(element1, { name: 'FadeOut' });
  animation.animate(element2, { name: 'ZoomOut' });
}, [])
  return (
    <div id="container">
      <div id="element1" ref={ele} => { element1 = ele; }></div>
      <div id="element2" ref={ele} => { element2 = ele; }></div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
{% enddraw %}
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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="index.css" rel="stylesheet" />
  <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://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>
</head>
<body>
  <div id='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

Enable or disable animation globally

Enable or disable animation for all React components globally by using the `setGlobalAnimation` method with one of the below options:

- `GlobalAnimationMode.Enable` - Enables the animation for all components, regardless of the individual component's animation settings.
- `GlobalAnimationMode.Disable` - Disables the animation for all components, regardless of the individual component's animation settings.
- `GlobalAnimationMode.Default` - Animation is enabled or disabled based on the component's animation settings.

In the below code snippet, animation is disabled.

~/SRC/INDEX.JS

```
import { GlobalAnimationMode, setGlobalAnimation } from "@syncfusion/ej2-base";
setGlobalAnimation(GlobalAnimationMode.Disable);
```

`setGlobalAnimation` method controls script-level animations only, and it is not applicable for direct CSS-level animations (animations defined from CSS classes or properties).

Drag and Drop in React

Drag and drop is a feature of a user interface that allows users to select an item or items and then move them to a different location or onto another interface element by "dragging" the selected item(s) with a pointing device (such as a mouse) and then "dropping" them at the desired location.

Syncfusion React components support drag and drop feature through two libraries. These are [Draggable](#) and [Droppable](#).

Draggable

The Syncfusion's [Draggable](#) library allows users to make any DOM element draggable by passing it as a parameter to the `Draggable` constructor. This can be useful for creating interactive and user-friendly interfaces, such as allowing a user to reorder items in a list by dragging them. The below code snippet enables the draggable functionality for the specific DOM element passed to the `Draggable` constructor.

INDEX.JSX

```
import * as React from 'react';
import { useEffect } from 'react';
import * as ReactDOM from 'react-dom';
import { Draggable } from '@syncfusion/ej2-base';
function App() {
  useEffect(() => {
    let draggable = new Draggable(document.getElementById('element'), {
      clone: false });
  }, []);
  return (<div id='container'>
    <div id='element'><p>Draggable Element</p></div>
  </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import * as React from 'react';
import { useEffect } from 'react';
import * as ReactDOM from 'react-dom';
import { Draggable } from '@syncfusion/ej2-base';
function App() {
    useEffect(() => {
        let draggable: Draggable = new
        Draggable(document.getElementById('element'), { clone: false });
    }, [])
    return (
        <div id='container'>
            <div id='element'><p>Draggable Element</p></div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Button</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="index.css" rel="stylesheet" />
    <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://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>
</head>
<body>
    <div id='sample'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>
```


Clone draggable element

Syncfusion provides the option to create a clone of a draggable element while the user is dragging it. It can be achieved by setting the [clone](#) property to `true`. Here's an example of how to create a clone of a draggable element.

INDEX.JSX

```
import * as React from 'react';
import { useEffect } from 'react';
import * as ReactDOM from 'react-dom';
import { Draggable } from '@syncfusion/ej2-base';
function App() {
    useEffect(() => {
        let draggable = new Draggable(document.getElementById('element'), {
clone: true });
    }, []);
    return (<div id='container'>
        <div id='element'><p>Draggable Element</p></div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import * as React from 'react';
import { useEffect } from 'react';
import * as ReactDOM from 'react-dom';
import { Draggable } from '@syncfusion/ej2-base';
function App() {
    useEffect(() => {
        let draggable: Draggable = new
Draggable(document.getElementById('element'), { clone: true });
    }, [])
    return (
        <div id='container'>
            <div id='element'><p>Draggable Element</p></div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Button</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="index.css" rel="stylesheet" />
```

```

    <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://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>
</head>
<body>
    <div id='sample'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Drag area

A drag area is a specific area within a user interface that is designated for drag and drop operations. When an object or element is dragged within a drag area, certain actions or events may be triggered. The user can specify the drag area by using the [dragArea](#) property. Refer to the below sample.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect } from 'react';
import { Draggable } from '@syncfusion/ej2-base';
function App() {
    useEffect(() => {
        let draggable = new Draggable(document.getElementById('draggable'), {
clone: false, dragArea: "#droppable" });
        }, []);
    return <div id='container'>
        <div id='droppable'><p>Drag Area </p></div>
        <div id='draggable'><p id='drag'>Draggable Element </p></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 { useEffect } from 'react';
import { Draggable } from '@syncfusion/ej2-base';
function App() {
    useEffect(() => {
        let draggable: Draggable = new
Draggable(document.getElementById('draggable'),{ clone: false, dragArea:
"#droppable" });
        }, [])

```

```

    return (
      <div id='container'>
        <div id='droppable'><p>Drag Area </p></div>
        <div id='draggable'><p id='drag'>Draggable Element </p></div>
      </div>
    );
  }
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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="index.css" rel="stylesheet" />
  <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://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>
</head>
<body>
  <div id='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Draggable

Draggable component refers to an area of a user interface that can receive a draggable component that is being moved by a user. Syncfusion's [Draggable](#) library converts any DOM element into a draggable zone, which accepts draggable elements.

When a draggable component is moved over a draggable component, the [drag](#) event can be triggered. The user can get details about the dropped element through the event argument. Based on the event argument, the user can append the dragged element to the target element.

Refer to the following code snippet to enable draggable zones.

INDEX.JSX

```

import * as React from 'react';

```

```
import * as ReactDOM from 'react-dom';
import { useEffect } from 'react';
import { Draggable, Droppable } from '@syncfusion/ej2-base';
function App() {
    useEffect(() => {
        let draggable = new Draggable(document.getElementById('draggable'), {
            clone: false });
        let droppable = new Droppable(document.getElementById('droppable'), {
            drop: (e) => {
                e.droppedElement.querySelector('#drag').textContent =
                'Dropped';
            }
        });
    }, []);
    return (<div id='container'>
        <div id='droppable'><p>Droppable Target </p></div>
        <div id='draggable'><p id='drag'>Draggable Element </p></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 { useEffect } from 'react';
import { Draggable, Droppable, DropEventArgs } from '@syncfusion/ej2-base';
function App() {
    useEffect(() => {
        let draggable: Draggable = new
        Draggable(document.getElementById('draggable'), { clone: false });
        let droppable: Droppable = new
        Droppable(document.getElementById('droppable'), {
            drop: (e: DropEventArgs) => {
                e.droppedElement.querySelector('#drag').textContent =
                'Dropped';
            }
        });
    }, [])
    return (
        <div id='container'>
            <div id='droppable'><p>Droppable Target </p></div>
            <div id='draggable'><p id='drag'>Draggable Element </p></div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Button</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="index.css" rel="stylesheet" />
<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://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>
</head>
<body>
    <div id='sample'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

Templates in Syncfusion React Components

Syncfusion React components are rendered with a pre-defined layout or structure that is used to define how the component should be rendered on the user interface. The user wants to customise the appearance of the component and add functionality that is specific to the needs of the application. Syncfusion React components have the option to achieve this using template support.

The user can use the JavaScript function to add custom template content to the Syncfusion React components. The JavaScript function is the simplest way to define the React components. This function accepts a "props" object argument with data and returns a React element. Refer to the below code snippet to create the template content using JavaScript function.

```

、
function gridTemplate(props) {
return (<div className='custom'>
<ButtonComponent>{props.ShipCountry}</ButtonComponent>
</div>);
}
、

```

Here, the JavaScript function name (gridTemplate) is assigned to the `template` property of the Grid component. Refer to the following code snippet.

```

`js
import './App.css';
import * as React from 'react';

```

```
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
import { ColumnDirective, ColumnsDirective, GridComponent } from '@syncfusion/ej2-react-grids';
export default function App() {
  const data = [
    { OrderID: 10248, CustomerID: 'VINET', ShipCountry: 'France' },
    { OrderID: 10249, CustomerID: 'TOMSP', ShipCountry: 'Germany' },
    { OrderID: 10250, CustomerID: 'HANAR', ShipCountry: 'Brazil' }
  ];
  function gridTemplate(props) {
    return (<div className='custom'>
      <ButtonComponent>{props.ShipCountry}</ButtonComponent>
    </div>);
  }
  return (<GridComponent dataSource={data}>
    <ColumnsDirective>
      <ColumnDirective field='OrderID' width='100' />
      <ColumnDirective field='CustomerID' width='100' />
      <ColumnDirective field='ShipCountry' width='100' template={gridTemplate} />
    </ColumnsDirective>
  </GridComponent>);
};
```

Stateless template

In React, a state is an object that contains data or information about the component. The component state can be used in React component templates to determine a component's behavior and render information to the user. The state can change based on user input, data from a server, the result of a calculation, or system-generated events.

Whenever the state changes, the component will automatically re-render to display the updated information to the user. This allows for a dynamic and interactive user experience.

For specific needs of the application, users don't want to re-render components during state updates. This can be achieved using the `statelessTemplates` property. It specifies the array of template names where the state value updates need to be ignored. It will increase the performance of the components if users prevent state updates for the templates that are not required. Refer to the following code snippet.

```
<TreeViewComponent fields={fields} statelessTemplates={['nodeTemplate']}
nodeTemplate={nodeTemplate} />
```

If the templates are defined in nested directives of the component, then pass the `statelessTemplates` property array value as "directiveTemplates" instead of the template names. Refer to the following code snippet.

```
<GridComponent dataSource={siteCedarData} statelessTemplates={['directiveTemplates']}>
  <ColumnsDirective>
    <ColumnDirective field="name" headerText="asset" textAlign="Left" />
    <ColumnDirective field="status" headerText="status" textAlign="Center" template={renderStatusCell} />
  </ColumnsDirective>
</GridComponent>
```

Syncfusion React components - Security

Security is a critical aspect of web applications to protect them from various threats and vulnerabilities. Using HTTPS for data encryption, validating and sanitizing user inputs, and implementing strong authentication measures such as multi-factor authentication are indispensable practices in Web application development.

Syncfusion React components are implemented with these security considerations.

Security Vulnerabilities

Security vulnerabilities in web applications refer to weaknesses or flaws in the design, implementation, or configuration of a web application that can be exploited by attackers to compromise the application's integrity, confidentiality, or availability. Here you can see some of the vulnerabilities.

- [Cross-Site Scripting](#) - XSS is a security vulnerability that can occur in web applications. These scripts can steal session cookies, redirect users to malicious websites, or deface the website. XSS vulnerabilities typically arise when the application fails to properly validate or encode user-supplied input before rendering it in the browser.
- [Cross-Site Request Forgery](#) - CSRF is a type of web security vulnerability that allows an attacker to force a logged-in user to perform actions on a web application without their consent or knowledge. CSRF attacks exploit the trust that a web application has in the user's browser by tricking it into sending unauthorized requests to the vulnerable application.
- **Injection Attacks** - These occur when an attacker injects malicious code (such as SQL injection, XML injection, or command injection) into input fields or parameters of a web application. If the application does not properly sanitize or validate user inputs, it can execute unintended commands or gain unauthorized access to sensitive data.

Syncfusion React components provide support for implementing web applications with enhanced security features.

Security Considerations

Security holds significant importance in software development, and the incorporation of security measures from the outset of the development process is vital for the protection of applications. Syncfusion takes a thorough approach to security in the development of React components,

encompassing all critical aspects. The following considerations provide a comprehensive overview of security measures.

- [Content Security Policy](#)
- [HTML Sanitizer](#)
- [Browser Storage](#)

Content Security Policy

[Content Security Policy](#) (CSP) is a one of the security feature, that helps the detect the cross-site-scripting(XSS) attacks and malicious code injection. It will throw the errors and warnings while using the inline-styles and inline scripts, eval, new Function, etc in your applications.

To implement Content Security Policy (CSP) in your application, include a `<meta>` tag with specified CSP directives. Syncfusion React components have been designed and implemented with adherence to these CSP directives, ensuring enhanced security. These directives are below.

CSP Directives

The following directives are essential for utilizing Syncfusion React components.

Directives	Description	Examples
<code>style-src</code>	Defines the allowed sources for loading stylesheets. This helps mitigate style-based attacks by restricting the locations from which styles can be applied.	<code>style-src 'self' https://cdn.syncfusion.com/ https://fonts.googleapis.com/ 'unsafe-inline';</code>
<code>font-src</code>	Defines the allowed sources for loading fonts. It helps prevent font-related security issues by restricting the locations from which fonts can be loaded.	<code>font-src 'self' https://fonts.googleapis.com/ https://fonts.gstatic.com/ data:cdn.syncfusion.com 'unsafe-inline';</code>
<code>img-src</code>	Specifies the allowed sources for loading images. It helps control from where images can be displayed on the web page.	<code>img-src 'self' data:"</code>

Utilizing a web worker within the Spreadsheet Control for exporting necessitates the addition of a specific directive to ensure proper functionality during the export process.

```
worker-src 'self' 'unsafe-inline' * blob;;
```

CSP Sources

The following sources refer to the origins from which resources such as styles, images, fonts are allowed to be loaded and executed on a web page.

Source	Description	Examples
<code>self</code>	Refers to the origin from which the protected document is being served, including the same URL scheme and port number	<code>style-src 'self'</code>
<code>data</code>	Enables a website to fetch resources using the data scheme, such as loading Base64-encoded images.	<code>img-src 'self' data:</code>

| `unsafe-inline` | Allows the use of inline resources, such as inline `style` elements. | `style-src 'self' https://fonts.googleapis.com/ 'unsafe-inline'` |

To know more information about the CSP, refer this [documentation](#).

HTML Sanitizer

An HTML sanitizer is a tool or program that helps remove potentially malicious or harmful code from HTML documents. This type of sanitizer is commonly used in web applications to prevent cross-site scripting (XSS) attacks, which can inject malicious code into a website and compromise user data. HTML sanitizers typically work by analyzing HTML code and removing any potentially dangerous or unwanted elements, such as script tags, inline styles, or event handlers. Other aspects of the HTML may also be modified or cleaned up, such as removing extra whitespace or fixing malformed code.

To avoid the risk of code injection, Syncfusion has provided the [enableHtmlSanitizer](#) API into its UI controls. This ensures that HTML strings submitted by users are sanitized, enhancing security measures against potential threats.

When this property is enabled, the HTML string undergoes a thorough sanitization process before being rendered in the component. This approach ensures that user inputs containing potential security threats are meticulously filtered, addressing the risk of XSS and contributing to the overall security robustness of our components in the face of potential attacks.

To sanitize input values in a web application using Syncfusion sanitizer, you can use the following code.

```
`js
import { SanitizeHtmlHelper } from '@syncfusion/ej2-base';

let html = '<script>alert("XSS");</script>';

let sanitizedHtml = SanitizeHtmlHelper.sanitize(html);
`
```

For sanitizing the template content using Syncfusion React components, please see the provided code below.

```
`ts
import { DialogComponent } from '@syncfusion/ej2-react-popups';
import * as React from 'react';
import { SanitizeHtmlHelper } from '@syncfusion/ej2-base';

function App() {
  return (
    <div className="App" id="dialog-target">
      <DialogComponent
        width="250px"
        enableHtmlSanitizer={false}
        target="#dialog-target"
        showCloseIcon={true}
      />
    </div>
  );
}
```

```
header={SanitizeHtmlHelper.sanitize('<div id="dlg-template" title="Nancy" class="e-icon-settings">
Nancy <div><div onmousemove=function(){alert("XSS")}>XSS</div>')}
closeOnEscape={false}
```

This is a dialog with header{' '}

```
</DialogComponent>
```

```
</div>
```

```
);
```

```
}
```

```
export default App;
```

```
,
```

INDEX.JSX

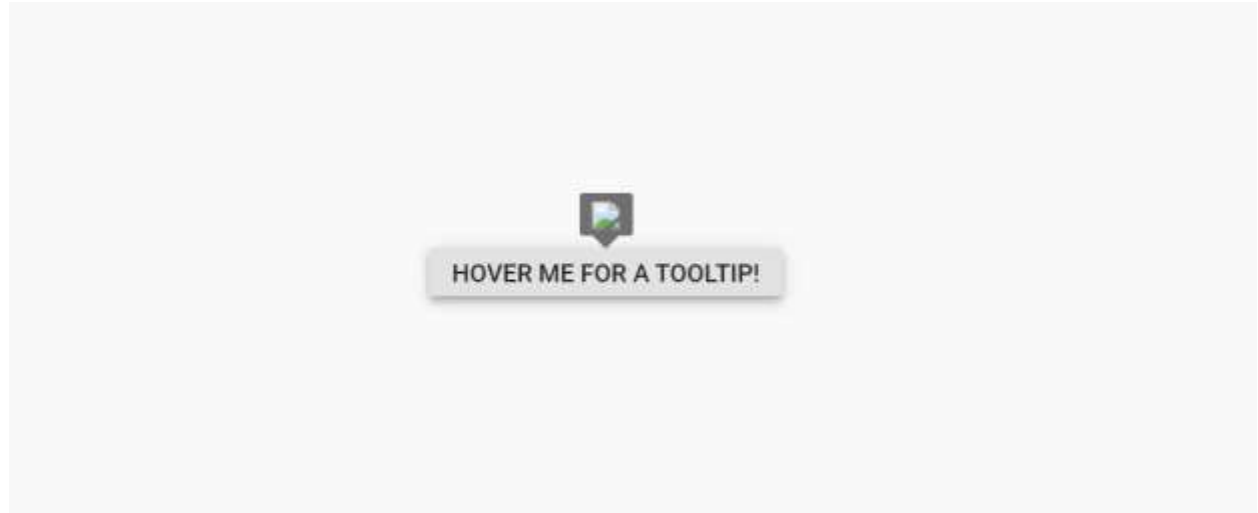
```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { TooltipComponent } from '@syncfusion/ej2-react-popups';
import './App.css';
function App() {
    return (<div id="container">
        <TooltipComponent position="TopCenter"
            content='<img src=text onerror=alert("XSS_Script_Attack") \/>'
            target="#target"
            enableHtmlSanitizer= {true}>
            <button className="e-btn tooltipElement" id="target" >Hover me for a
tooltip!</button>
        </TooltipComponent>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

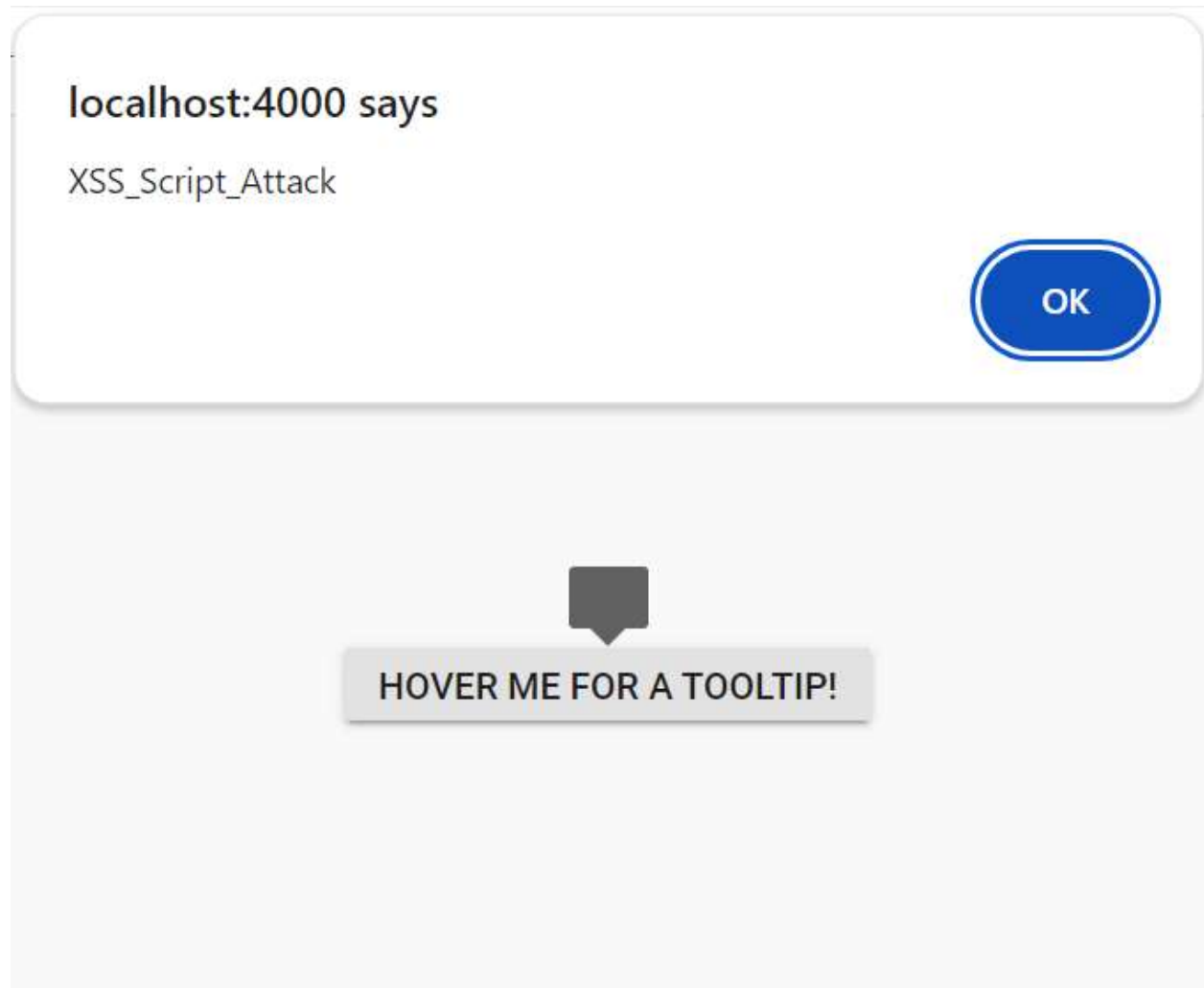
```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { TooltipComponent } from '@syncfusion/ej2-react-popups';
import './App.css';
function App() {
    return (<div id="container">
        <TooltipComponent position="TopCenter"
            content='<img src=text onerror=alert("XSS_Script_Attack") \/>'
            target="#target"
            enableHtmlSanitizer= {true}>
            <button className="e-btn tooltipElement" id="target" >Hover me for a
tooltip!</button>
        </TooltipComponent>
    </div>);
}
```

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

When `enableHtmlSanitizer` is `true`, the content will be sanitized and displays the code.



When `enableHtmlSanitizer` is `false` or not included this property, the malicious code will be interpreted as script, and the alert pop-up window will be open.



[Browser Storage](#)

Browser storage refers to the mechanisms provided by web browsers to store data locally on a user's device. Syncfusion React components utilize the following storage options only.

- Local Storage

[Local Storage](#)

[Local Storage](#) is a type of web storage mechanism provided by web browsers that allows web applications to store data locally on a user's device. It provides a simple key-value pair storage interface and is accessible via React.

Syncfusion React components utilize local storage only when persistence is enabled.

[Globalization](#)

[Globalization](#)

Globalization is the combination of adapting the control to various languages by parsing and formatting the date or numbers (Internationalization (L18N)) and adding cultural-specific customizations and translating the text (Localization (L10N)). The Syncfusion React UI components are specific to the **American English (en-US)** culture by default.

The globalization in Syncfusion React UI is enabled by

- [Localization](#) - It allows you to localize the text content of the Syncfusion React UI Components.
- [Internationalization](#) - It provides support for formatting and parsing date and number objects.

Internationalization

The **Internationalization** library provides support for formatting and parsing date and number objects using the official [Unicode CLDR](#) JSON data. The **en-US** locale is set as default *culture* and **USD** is set as default *currencyCode* for all Syncfusion React UI Components.

Loading Culture Data

It requires the following CLDR data to be load using **loadCldr** function for cultures other than **en-US**.

File Name	Path
ca-gregorian	cldr/main/en/ca-gregorian.json
timeZoneNames	cldr/main/en/timeZoneNames.json
numbers	cldr/main/en/numbers.json
numberingSystems	cldr/supplemental/numberingSystems.json
currencies	cldr/main/en/currencies.json

Note: For **en**, dependency files are already loaded in the library.

Installing CLDR Data

CLDR data is available as npm package. So, we can install it through below command to our package.

```
`bash
npm install cldr-data
`
```

Binding to i18n library

```
`ts
import { loadCldr } from '@syncfusion/ej2-base';
loadCldr(enNumberData, entimeZoneData);
`
```

Changing culture and currency code

To set the default culture and the currency code for all Syncfusion React UI Components, you can use the methods **setCulture** for setting the default locale and **setCurrencyCode** for setting the currency code.

```
`ts
import { setCulture, setCurrencyCode } from '@syncfusion/ej2-base';
setCulture('ar');
setCurrencyCode('QAR');
```

Note: If global culture is not set, then **en-US** is set as the default locale, and **USD** is set as the default currency code.

Manipulating Numbers

Supported format string

Based on the [NumberFormatOptions](#) number formatting and parsing operations are processed. You need to specify some or all of the following properties mentioned below table.

No	Properties	Description
1	format	Denotes the format to be set .Possible values are 1. N - denotes numeric type 2. C - denotes currency type 3. P – denotes percentage type. Ex: <code>formatNumber(1234344, {format:'N4'})</code> . > Note: If no format is specified it takes numeric as default format type.
2	minimumFractionDigits	Indicates the minimum number of fraction digits . Possible values are 0 to 20.
3	maximumFractionDigits	Indicates the maximum number of fraction digits. Possible values are 0 to 20.
4	minimumSignificantDigits	Indicates the minimum number of significant digits. Possible values are 1 to 21. > Note: If minimumSignificantDigits is given it is mandatory to give maximumSignificantDigits
5	maximumSignificantDigits	Indicates the maximum number of significant digits. . Possible values are 1 to 21. > Note: If maximumSignificantDigits is given it is mandatory to give minimumSignificantDigits
6	useGrouping	Indicates whether to enable the group separator or not . By default grouping value will be true.
7	minimumIntegerDigits	Indicates the minimum number of the integer digits to be placed in the value. Possible values are 1 to 21.
8	currency	Indicates the currency code which needs to be considered for the currency formatting.

Note: The **minimumIntegerDigits**, **minimumFractionDigits** and **maximumFractionDigits** are categorized as group one, **minimumSignificantDigits** and **maximumSignificantDigits** are categorized as group two.

If group two properties are defined, then the group one property will be ignored.

Custom number formatting and parsing

Custom number formatting and parsing can also be achieved by directly specifying the pattern in the **format** property of **NumberFormatOptions**. One or more of the custom format specifiers listed in the table below can be used to create custom number format.

Specifier	Description	Input	Format Output
-----	-----	-----	-----

| 0 | Replaces the zero with the corresponding digit if one is present. Otherwise, zero appears in the result string. | `instance.formatNumber(123,{format: '0000' })` | '0123' |

| # | Replaces the "#" symbol with the corresponding digit if one is present; otherwise, no digit appears in the result string. | `instance.formatNumber(1234,{format: '####' })` | '1234' |

| . | Denotes the number of digits allowed after the decimal points if it's not specified then no need to specify decimal point values. | `instance.formatNumber(546321,{format: '###0.##0#' })` | '546321.000' |

| % | Percent specifier denotes the percentage type format. | `instance.formatNumber(1,{format: '0000 %' })` | '0100 %' |

| \$ | Denotes the currency type format based on the global currency code specified. | `instance.formatNumber(13,{format: '$ ###.00' })`; | '\$ 13.00' |

| ; | Denotes separate formats for positive, negative and zero values. | `instance.formatNumber(-120,{format: '###.##;(###.00);-0'})`; | '(120.00)' |

| 'String' (single Quotes) | Denotes the characters enclosed within single Quote(') to be replaced in the resultant string. | `instance.formatNumber(-123.44,{format: "####.## '@'"})` | '123.44 @' |

Note: If a custom format pattern is specified, other [NumberFormatOptions](#) properties will not be considered.

Number parsing

``getNumberParser``

The [getNumberParser](#) method, which will return a function that parses a given string based on the [NumberFormatOptions](#) specified.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect } from 'react';
import { Internationalization } from '@syncfusion/ej2-base';
function App() {
  useEffect(() => {
    let intl = new Internationalization();
    let nParser = intl.getNumberParser({ format: 'P2', useGrouping: false
  });

  let val = nParser('123567.45%');
  document.querySelector('.result').innerHTML = val + '';
}, []);
  return (<div id="container">
    <div>FormattedValue:<span className="format
text">123567.45%</span></div>
    <div>ParsedOutput:<span className="result text"/></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 { useEffect } from 'react';
import { Internationalization } from '@syncfusion/ej2-base';
function App() {
  useEffect(() => {
    let intl: Internationalization = new Internationalization();
    let nParser: Function = intl.getNumberParser({ format: 'P2' ,
useGrouping: false});
    let val: number = nParser('123567.45%');
    document.querySelector('.result').innerHTML = val + ' ';
  }, [])
  return (
    <div id="container">
      <div>FormattedValue:<span className="format
text">123567.45%</span></div>
      <div>ParsedOutput:<span className="result text"/></div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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-
grids/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://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='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

``parseNumber``

The [parseNumber](#) method, which takes two arguments, the string value and [NumberFormatOptions](#) and returns the numeric value.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect } from 'react';
import { Internationalization } from '@syncfusion/ej2-base';
function App() {
  useEffect(() => {
    let intl = new Internationalization();
    let val = intl.parseNumber('$01,234,545.650', { format: 'C3',
currency: 'USD', minimumIntegerDigits: 8 });
    document.querySelector('.result').innerHTML = val + '';
  }, []);
  return (<div id="container">
    <div>FormattedValue:<span className="format
text">$01,234,545.650</span></div>
    <div>ParsedOutput:<span className="result text"/></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 { useEffect } from 'react';
import { Internationalization } from '@syncfusion/ej2-base';
function App() {
  useEffect(() => {
    let intl: Internationalization = new Internationalization();
    let val: number = intl.parseNumber('$01,234,545.650', { format: 'C3',
currency: 'USD', minimumIntegerDigits: 8 });
    document.querySelector('.result').innerHTML = val + '';
  }, []);
  return (
    <div id="container">
      <div>FormattedValue:<span className="format
text">$01,234,545.650</span></div>
      <div>ParsedOutput:<span className="result text" /></div>
    </div>
  );
}
```

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

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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-
grids/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://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='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

Number formatting

`getNumberFormat`

The [getNumberFormat](#) method will return a function that formats a given number based on the [NumberFormatOptions](#) specified.

INDEX.JSX

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect } from 'react';
import { Internationalization } from '@syncfusion/ej2-base';
```

```
function App() {
  useEffect(() => {
    let intl = new Internationalization();
    let nFormatter = intl.getNumberFormat({ skeleton: 'C3', currency:
'USD', minimumIntegerDigits: 8 });
    let formattedValue = nFormatter(1234545.65);
    document.querySelector('.result').innerHTML = formattedValue;
  }, []);
  return (<div id="container">
    <div>Value:<span className="format text">1234545.65</span></div>
    <div>Formatted Value:<span className="result text"/></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 { useEffect } from 'react';
import { Internationalization } from '@syncfusion/ej2-base';
function App() {
  useEffect(() => {
    let intl: Internationalization = new Internationalization();
    let nFormatter: Function = intl.getNumberFormat({ skeleton: 'C3',
currency: 'USD', minimumIntegerDigits: 8 });
    let formattedValue: string = nFormatter(1234545.65);
    document.querySelector('.result').innerHTML = formattedValue;
  }, [])
  return (
    <div id="container">
      <div>Value:<span className="format text">1234545.65</span></div>
      <div>Formatted Value:<span className="result text" /></div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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-
grids/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://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='sample'>
        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

`formatNumber`

The [formatNumber](#) method, which takes two arguments, a numeric value and [NumberFormatOptions](#) and returns the formatted string.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect } from 'react';
import { Internationalization } from '@syncfusion/ej2-base';
function App() {
    useEffect(() => {
        let intl = new Internationalization();
        let formattedString = intl.formatNumber(12345.65, {
            format: 'C5', useGrouping: false,
            minimumSignificantDigits: 1, maximumSignificantDigits: 3
        });
        document.querySelector('.result').innerHTML = formattedString;
    }, []);
    return (<div id="container">
        <div>Value:<span className="format text">12345.65</span></div>
        <div>Formatted Value:<span className="result text"/></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 { useEffect } from 'react';
import { Internationalization } from '@syncfusion/ej2-base';
function App() {
  useEffect(() => {
    let intl: Internationalization = new Internationalization();
    let formattedString: string = intl.formatNumber(12345.65, {
      format: 'C5', useGrouping: false,
      minimumSignificantDigits: 1, maximumSignificantDigits: 3
    });
    document.querySelector('.result').innerHTML = formattedString;
  }, [])
  return (
    <div id="container">
      <div>Value:<span className="format text">12345.65</span></div>
      <div>Formatted Value:<span className="result text" /></div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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-
grids/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://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='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Manipulating DateTime

Supported format string

Date formatting and parsing operations are performed based on the [DateFormatOptions](#). You need to specify some or all of the following properties mentioned in the table below.

Options	Descriptions
---	---
Type	It specifies the type of format to be used supported types . 1. date 2. dateTime 3. time Based on the type specified the supported skeletons are given below 1. short 2. medium, 3. long 4. full Ex: <code>formatDate(new Date(), {type: 'date', skeleton:medium})</code> > Note: If not type specified then date type is set by default
skeleton	Specifies the format in which the dateTime format will process

<!-- markdownlint-disable MD036 -->

Date type skeletons

skeleton	Option input	Format Output
---	---	---
short	{type: 'date', skeleton:'short'}	11/4/16
medium	{type: 'date', skeleton:'medium'}	Nov 4, 2016
long	{type: 'date', skeleton:'long'}	November 4, 2016
full	{type: 'date', skeleton:full}	Friday, November 4, 2016

Time type skeletons

skeleton	Option input	Format Output
---	---	---
short	{type: 'time', skeleton:'short'}	1:03 PM
medium	{type: 'time', skeleton:'medium'}	1:03:04 PM
Long	{type: 'time', skeleton:'long'}	1:03:04 PM GMT+5
full	{type: 'time', skeleton:'full'}	1:03:04 PM GMT+05:30

DateTime type skeletons

Skeleton	Option input	Format Output
---	---	---
short	{type: 'dateTime', skeleton:'short'}	11/4/16, 1:03 PM

medium	{type: 'dateTime', skeleton:'medium'}	Nov 4, 2016, 1:03:04 PM
Long	{type: 'dateTime', skeleton:'long'}}	November 4, 2016 at 1:03:04 PM GMT+5
full	{type: 'dateTime', skeleton:'full'}}	Friday, November 4, 2016 at 1:03:04 PM GMT+05:30

Additional skeletons

Apart from the standard date type formats, additional formats are supported by using the additional skeletons given in the below table.

skeleton	Option input	Format Output
---	---	---
d	{skeleton:'d'}	7
E	{skeleton:'E'}	Mon
Ed	{skeleton:'Ed'}	7 Mon
Ehm	{skeleton:'Ehm'}	Mon 12:43 AM
EHm	{skeleton:'EHm'}}	Mon 12:43
Ehms	{skeleton:'Ehms' }	Mon 2:45:23 PM
EHms	{skeleton:'EHms'}}	Mon 12:45:45
Gy	{skeleton:'Gy' }	2016 AD
GyMMM	{skeleton:'GyMMM'}	: Nov 2016 AD
GyMMMd	{skeleton:'GyMMMd'}	Nov 7, 2016 AD
GyMMMEd	{skeleton:'GyMMMEd'}	Mon, Nov 7, 2016 AD
h	{skeleton:'h'}	12 PM
H	{skeleton:'H'}	12
hm	{skeleton:'hm'}	12:59 PM
Hm	{skeleton:'Hm'}	12:59
hms	{skeleton:'hms'}	12:59:13 PM
Hms	{skeleton:'Hms'}	12:59:13
M	{skeleton:'M'}	11
Md	{skeleton:'Md'}	11/7
MEd	{skeleton:'hms'}	Mon, 11/7
MMM	{skeleton:'MMM'}	Nov
MMMEd	{skeleton:'MMMEd'}	Mon, Nov 7
MMMd	{skeleton:'MMMEd'}	Nov 7
ms	{skeleton:'ms'}	59:13
y	{skeleton:'y' }	2016

```
| yM | {skeleton:'yM' } | 11/2016 |
| yMd | {skeleton:'yMd' } | 11/7/2016 |
| yMEd | {skeleton:'yMEd' } | Mon, 11/7/2016 |
| yMMM | {skeleton:'yMMM' } | Nov 2016 |
| yMMMd | {skeleton:'yMMMd' } | Nov 7, 2016 |
| yMMMEd | {skeleton:'yMMMEd' } | Mon, Nov 7, 2016 |
| yMMM | {skeleton:'yMMM' } | Nov 2016 |
```

Note: Culture specific format skeletons are also supported.

Custom formats

To use the custom date and time formats, we need to specify the date/time pattern directly in the *format* property.

A custom format string must contain one or more of the following standard date/time symbols.

Symbols	Description
-----	Denotes the era in the date
G	Denotes the year.
y	Denotes month.
M / L	Denotes the day of week.
E / c	Denotes the day of month.
d	Denotes the hour. <i>h</i> for 12 hour and <i>H</i> for 24 hours format.
h / H	Denotes minutes.
m	Denotes seconds.
s	Denotes the am/pm designator it will only be displayed if hour is specified in the h format.
a	Denotes the time zone.
z	To display words in the formatted date you can specify the words with in the single quotes

Custom format example

```
`ts
import { Internationalization } from '@syncfusion/ej2-base';
let intl: Internationalization = new Internationalization();
let formattedString: string = intl.formatDate(new Date('1/12/2014 10:20:33'), { format: '\year:\y'
\month:\ MM' });
//Output: "year:2014 month:01"
```


Note: If the format property is given as an option, other properties are not considered.

<!-- markdownlint-enable MD036 -->

Date parsing

`getDateParser`

The [getDateParser](#) method will return a function that parses a given string based on the [DateFormatOptions](#) specified.

INDEX.JSX

```
import { Internationalization } from '@syncfusion/ej2-base';
import * as React from 'react';
import { useEffect } from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    useEffect(() => {
        let intl = new Internationalization();
        let dParser = intl.getDateParser({ skeleton: 'full', type: 'dateTime'
    });
        let val = dParser('Friday, November 4, 2016 at 1:03:04 PM
GMT+05:30');
        document.querySelector('.result').innerHTML = val.toString();
    }, []);
    return (<div id="container">
        <div>Formatted value:<span className="format text">Friday, November 4,
2016 at 1:03:04 PM GMT+05:30</span></div>
        <div>parsed Value:<span className="result text"/></div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.TSX

```
import { Internationalization } from '@syncfusion/ej2-base';
import * as React from 'react';
import { useEffect } from 'react';
import * as ReactDOM from 'react-dom';
function App() {
    useEffect(() => {
        let intl: Internationalization = new Internationalization();
        let dParser: Function = intl.getDateParser({ skeleton: 'full', type:
'dateTime' });
        let val: Date = dParser('Friday, November 4, 2016 at 1:03:04 PM
GMT+05:30');
        document.querySelector('.result').innerHTML = val.toString();
    }, [])
    return (
        <div id="container">
            <div>Formatted value:<span className="format text">Friday, November 4,
2016 at 1:03:04 PM GMT+05:30</span></div>
            <div>parsed Value:<span className="result text" /></div>
        </div>
    );
}
```

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

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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-
grids/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://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='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>
```

``parseDate``

The date object is returned by the [parseDate](#) method, which takes two arguments, a string value and [DateFormatOptions](#).

INDEX.JSX

```
import { Internationalization } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect } from 'react';
function App() {
```

```

    useEffect(() => {
        let intl = new Internationalization();
        let val = intl.parseDate('11/2016', { skeleton: 'yM' });
        document.querySelector('.result').innerHTML = val.toString();
    }, []);
    return (<div id="container">
        <div>Formatted value:<span className="format text">11/2016</span></div>
        <div>parsed Value:<span className="result text"/></div>
    </div>);
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.TSX

```

import { Internationalization } from '@syncfusion/ej2-base';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect } from 'react';
function App() {
    useEffect(() => {
        let intl: Internationalization = new Internationalization();
        let val: Date = intl.parseDate('11/2016', { skeleton: 'yM' });
        document.querySelector('.result').innerHTML = val.toString();
    }, [])
    return (
        <div id="container">
            <div>Formatted value:<span className="format text">11/2016</span></div>
            <div>parsed Value:<span className="result text" /></div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion React Button</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-grids/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://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='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Date formatting

``getDateFormat``

The [getDateFormat](#) method, which will return a function that formats a given date object based on the [DateFormatOptions](#) specified.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect } from 'react';
import { Internationalization } from '@syncfusion/ej2-base';
function App() {
  useEffect(() => {
    let intl = new Internationalization();
    let dFormatter = intl.getDateFormat({ skeleton: 'full', type:
'dateTime' });
    let formattedString = dFormatter(new Date('1/12/2014 10:20:33'));
    document.querySelector('.result').innerHTML = formattedString;
  }, []);
  return (
    <div id="container">
      <div>DateValue:<span className="format text">new Date('1/12/2014
10:20:33')</span></div>
      <div>Formatted Value:<span className="result text"/></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 { useEffect } from 'react';

```

```
import { Internationalization } from '@syncfusion/ej2-base';
function App() {
  useEffect(() => {
    let intl: Internationalization = new Internationalization();
    let dFormatter: Function = intl.getDateFormat({ skeleton: 'full', type:
'dateTime' });
    let formattedString: string = dFormatter(new Date('1/12/2014 10:20:33'));
    document.querySelector('.result').innerHTML = formattedString;
  }, [])
  return (
    <div id="container">
      <div>DateValue:<span className="format text">new Date('1/12/2014
10:20:33')</span></div>
      <div>Formatted Value:<span className="result text" /></div>
    </div>
  );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));
```

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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-
grids/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://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='sample'>
```

```

        <div id='loader'>Loading....</div>
    </div>
</body>
</html>

```

``formatDate``

The [formatDate](#) method, which takes two arguments, the date object and [DateFormatOptions](#), returns the formatted string.

INDEX.JSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { useEffect } from 'react';
import { Internationalization } from '@syncfusion/ej2-base';
function App() {
    useEffect(() => {
        let intl = new Internationalization();
        let formattedString = intl.formatDate(new Date('1/12/2014 10:20:33'),
    { skeleton: 'GyMMM' });
        document.querySelector('.result').innerHTML = formattedString;
    }, []);
    return (<div id="container">
        <div>DateValue:<span className="format text">new Date('1/12/2014
10:20:33')</span></div>
        <div>Formatted Value:<span className="result text"/></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 { useEffect } from 'react';
import { Internationalization } from '@syncfusion/ej2-base';
function App() {
    useEffect(() => {
        let intl: Internationalization = new Internationalization();
        let formattedString: string = intl.formatDate(new Date('1/12/2014
10:20:33'), { skeleton: 'GyMMM' });
        document.querySelector('.result').innerHTML = formattedString;
    }, [])
    return (
        <div id="container">
            <div>DateValue:<span className="format text">new Date('1/12/2014
10:20:33')</span></div>
            <div>Formatted Value:<span className="result text" /></div>
        </div>
    );
}
export default App;
ReactDOM.render(<App />, document.getElementById('sample'));

```

INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion React Button</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-
grids/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://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='sample'>
    <div id='loader'>Loading....</div>
  </div>
</body>
</html>

```

Getting started with Localization

Localization library allows you to localize the text content of the Syncfusion React UI Components. This is useful if you want to display the UI in a language other than English.

Loading translations

To load a translation object in your application, you can use the load function from the @syncfusion/ej2-base module. This function takes an object that contains the translations for various languages, with the keys being the language codes and the values being the translation objects.

INDEX.JSX

```

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

```

```

import { L10n } from '@syncfusion/ej2-base';
import { GridComponent, Inject, ColumnsDirective, ColumnDirective, Page,
Group } from '@syncfusion/ej2-react-grids';
import { data } from './datasource';
L10n.load({
    'de-DE': {
        'grid': {
            'EmptyRecord': 'Keine Aufzeichnungen angezeigt',
            'GroupDropArea': 'Ziehen Sie einen Spaltenkopf hier, um die
Gruppe ihre Spalte',
            'UnGroup': 'Klicken Sie hier, um die Gruppierung aufheben',
            'EmptyDataSourceError': 'DataSource darf bei der Erstaustauslastung
nicht leer sein, da Spalten aus der dataSource im AutoGenerate
Spaltenraster',
            'Item': 'Artikel',
            'Items': 'Artikel'
        },
        'pager': {
            'currentPageInfo': '{0} von {1} Seiten',
            'totalItemsInfo': '({0} Beiträge)',
            'firstPageTooltip': 'Zur ersten Seite',
            'lastPageTooltip': 'Zur letzten Seite',
            'nextPageTooltip': 'Zur nächsten Seite',
            'previousPageTooltip': 'Zurück zur letzten Seit',
            'nextPagerTooltip': 'Zum nächsten Pager',
            'previousPagerTooltip': 'Zum vorherigen Pager'
        }
    }
});
function App() {
    const pageOptions = { pageSize: 6 };
    return (<GridComponent dataSource={data} locale='de-DE'
allowPaging={true} pageSettings={pageOptions} allowGrouping={true}>
        <ColumnsDirective>
            <ColumnDirective field='OrderID' headerText='Order ID'
width='120' textAlign="Right"></ColumnDirective>
            <ColumnDirective field='CustomerID' headerText='Customer ID'
width='150'></ColumnDirective>
            <ColumnDirective field='ShipCity' headerText='Ship City'
width='150'></ColumnDirective>
            <ColumnDirective field='ShipName' headerText='Ship Name'
width='150'></ColumnDirective>
        </ColumnsDirective>
        <Inject services={[Page, Group]}></Inject>
    </GridComponent>);
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('grid'));

```

INDEX.TSX

```

import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { L10n } from '@syncfusion/ej2-base';

```



```

import { GridComponent, Inject, ColumnsDirective, ColumnDirective, Page,
Group } from '@syncfusion/ej2-react-grids';
import { data } from './datasource';
L10n.load({
  'de-DE': {
    'grid': {
      'EmptyRecord': 'Keine Aufzeichnungen angezeigt',
      'GroupDropArea': 'Ziehen Sie einen Spaltenkopf hier, um die
Gruppe ihre Spalte',
      'UnGroup': 'Klicken Sie hier, um die Gruppierung aufheben',
      'EmptyDataSourceError': 'DataSource darf bei der Erstausrüstung
nicht leer sein, da Spalten aus der dataSource im AutoGenerate
Spaltenraster',
      'Item': 'Artikel',
      'Items': 'Artikel'
    },
    'pager': {
      'currentPageInfo': '{0} von {1} Seiten',
      'totalItemsInfo': '({0} Beiträge)',
      'firstPageTooltip': 'Zur ersten Seite',
      'lastPageTooltip': 'Zur letzten Seite',
      'nextPageTooltip': 'Zur nächsten Seite',
      'previousPageTooltip': 'Zurück zur letzten Seite',
      'nextPagerTooltip': 'Zum nächsten Pager',
      'previousPagerTooltip': 'Zum vorherigen Pager'
    }
  }
});
function App() {
  const pageOptions : Object = { pageSize: 6 };
  return (<GridComponent dataSource={data} locale='de-DE'
allowPaging={true} pageSettings={pageOptions} allowGrouping={true}>
    <ColumnsDirective>
      <ColumnDirective field='OrderID' headerText='Order ID'
width='120' textAlign="Right"></ColumnDirective>
      <ColumnDirective field='CustomerID' headerText='Customer ID'
width='150'></ColumnDirective>
      <ColumnDirective field='ShipCity' headerText='Ship City'
width='150'></ColumnDirective>
      <ColumnDirective field='ShipName' headerText='Ship Name'
width='150'></ColumnDirective>
    </ColumnsDirective>
    <Inject services={[Page, Group]}/>
  </GridComponent>
);
export default App;
ReactDOM.render(<App />, document.getElementById('grid'));

```

Changing current locale

The current locale can be changed for all the Syncfusion React UI Components in your application by invoking `setCulture` function with your desired culture name.

`ts

```
import {L10n, setCulture} from '@syncfusion/ej2-base';
```

```

L10n.load({
'fr-BE': {
'Grid': {
'EmptyRecord': 'Aucun enregistrement à afficher',
'InvalidFilterMessage': 'Données de filtrage invalides',
'UnGroup': 'Cliquez ici pour dégroupier '
}
}
});
setCulture('fr-BE');

```

Note: Before changing a culture globally, you need to ensure that locale text for the concerned culture is loaded through `L10n.load` function.

State Persistence in Syncfusion React components

Syncfusion React UI components support persisting their state across page refreshes or navigation. To enable this feature, set the `enablePersistence` property to `true` for the desired component. This stores the component's state in the browser's `localStorage` object on the `unload` event of the page. For example, the `enablePersistence` property can be set for the Grid component, as shown in the following code snippet.

DATASOURCE.JSX

```

export let data = [
  {
    OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new
Date(8364186e5),
    ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims',
    ShipAddress: '59 rue de l Abbaye',
    ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France',
    Freight: 32.38, Verified: !0
  },
  {
    OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new
Date(836505e6),
    ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress:
'Luisenstr. 48',
    ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany',
    Freight: 11.61, Verified: !1
  },
  {
    OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new
Date(8367642e5),
    ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress:
'Rua do Paço, 67',
    ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil',
    Freight: 65.83, Verified: !0
  },

```

```

    {
      OrderID: 10251, CustomerID: 'VICTE', EmployeeID: 3, OrderDate: new
Date(8367642e5),
      ShipName: 'Victuailles en stock', ShipCity: 'Lyon', ShipAddress: '2,
rue du Commerce',
      ShipRegion: 'CJ', ShipPostalCode: '69004', ShipCountry: 'France',
      Freight: 41.34, Verified: !0
    },
    {
      OrderID: 10252, CustomerID: 'SUPRD', EmployeeID: 4, OrderDate: new
Date(8368506e5),
      ShipName: 'Suprêmes délices', ShipCity: 'Charleroi', ShipAddress:
'Boulevard Tirou, 255',
      ShipRegion: 'CJ', ShipPostalCode: 'B-6000', ShipCountry: 'Belgium',
      Freight: 51.3, Verified: !0
    },
    {
      OrderID: 10253, CustomerID: 'HANAR', EmployeeID: 3, OrderDate: new
Date(836937e6),
      ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress:
'Rua do Paço, 67',
      ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil',
      Freight: 58.17, Verified: !0
    },
    {
      OrderID: 10254, CustomerID: 'CHOPS', EmployeeID: 5, OrderDate: new
Date(8370234e5),
      ShipName: 'Chop-suey Chinese', ShipCity: 'Bern', ShipAddress:
'Hauptstr. 31',
      ShipRegion: 'CJ', ShipPostalCode: '3012', ShipCountry: 'Switzerland',
      Freight: 22.98, Verified: !1
    },
    {
      OrderID: 10255, CustomerID: 'RICSU', EmployeeID: 9, OrderDate: new
Date(8371098e5),
      ShipName: 'Richter Supermarkt', ShipCity: 'Genève', ShipAddress:
'Starenweg 5',
      ShipRegion: 'CJ', ShipPostalCode: '1204', ShipCountry: 'Switzerland',
      Freight: 148.33, Verified: !0
    },
    {
      OrderID: 10256, CustomerID: 'WELLI', EmployeeID: 3, OrderDate: new
Date(837369e6),
      ShipName: 'Wellington Importadora', ShipCity: 'Resende', ShipAddress:
'Rua do Mercado, 12',
      ShipRegion: 'SP', ShipPostalCode: '08737-363', ShipCountry: 'Brazil',
      Freight: 13.97, Verified: !1
    },
    {
      OrderID: 10257, CustomerID: 'HILAA', EmployeeID: 4, OrderDate: new
Date(8374554e5),
      ShipName: 'HILARION-Abastos', ShipCity: 'San Cristóbal', ShipAddress:
'Carrera 22 con Ave. Carlos Soublette #8-35',
      ShipRegion: 'Táchira', ShipPostalCode: '5022', ShipCountry:
'Venezuela', Freight: 81.91, Verified: !0
    },
    {

```

```

    OrderID: 10258, CustomerID: 'ERNSH', EmployeeID: 1, OrderDate: new
Date(8375418e5),
    ShipName: 'Ernst Handel', ShipCity: 'Graz', ShipAddress: 'Kirchgasse
6',
    ShipRegion: 'CJ', ShipPostalCode: '8010', ShipCountry: 'Austria',
Freight: 140.51, Verified: !0
    },
    {
        OrderID: 10259, CustomerID: 'CENTC', EmployeeID: 4, OrderDate: new
Date(8376282e5),
        ShipName: 'Centro comercial Moctezuma', ShipCity: 'México D.F.',
ShipAddress: 'Sierras de Granada 9993',
        ShipRegion: 'CJ', ShipPostalCode: '05022', ShipCountry: 'Mexico',
Freight: 3.25, Verified: !1
    },
    {
        OrderID: 10260, CustomerID: 'OTTIK', EmployeeID: 4, OrderDate: new
Date(8377146e5),
        ShipName: 'Ottilies Käseladen', ShipCity: 'Köln', ShipAddress:
'Mehrheimerstr. 369',
        ShipRegion: 'CJ', ShipPostalCode: '50739', ShipCountry: 'Germany',
Freight: 55.09, Verified: !0
    },
    {
        OrderID: 10261, CustomerID: 'QUEDE', EmployeeID: 4, OrderDate: new
Date(8377146e5),
        ShipName: 'Que Delícia', ShipCity: 'Rio de Janeiro', ShipAddress:
'Rua da Panificadora, 12',
        ShipRegion: 'RJ', ShipPostalCode: '02389-673', ShipCountry: 'Brazil',
Freight: 3.05, Verified: !1
    },
    {
        OrderID: 10262, CustomerID: 'RATTC', EmployeeID: 8, OrderDate: new
Date(8379738e5),
        ShipName: 'Rattlesnake Canyon Grocery', ShipCity: 'Albuquerque',
ShipAddress: '2817 Milton Dr.',
        ShipRegion: 'NM', ShipPostalCode: '87110', ShipCountry: 'USA',
Freight: 48.29, Verified: !0
    }
];

```

DATASOURCE.TSX

```

export let data: Object[] = [
    {
        OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new
Date(8364186e5),
        ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims',
ShipAddress: '59 rue de l Abbaye',
        ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France',
Freight: 32.38, Verified: !0
    },
    {
        OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new
Date(836505e6),

```

```

        ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress:
        'Luisenstr. 48',
        ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany',
        Freight: 11.61, Verified: !1
    },
    {
        OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new
        Date(8367642e5),
        ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress:
        'Rua do Paço, 67',
        ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil',
        Freight: 65.83, Verified: !0
    },
    {
        OrderID: 10251, CustomerID: 'VICTE', EmployeeID: 3, OrderDate: new
        Date(8367642e5),
        ShipName: 'Victuailles en stock', ShipCity: 'Lyon', ShipAddress: '2,
        rue du Commerce',
        ShipRegion: 'CJ', ShipPostalCode: '69004', ShipCountry: 'France',
        Freight: 41.34, Verified: !0
    },
    {
        OrderID: 10252, CustomerID: 'SUPRD', EmployeeID: 4, OrderDate: new
        Date(8368506e5),
        ShipName: 'Suprêmes délices', ShipCity: 'Charleroi', ShipAddress:
        'Boulevard Tirou, 255',
        ShipRegion: 'CJ', ShipPostalCode: 'B-6000', ShipCountry: 'Belgium',
        Freight: 51.3, Verified: !0
    },
    {
        OrderID: 10253, CustomerID: 'HANAR', EmployeeID: 3, OrderDate: new
        Date(836937e6),
        ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress:
        'Rua do Paço, 67',
        ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil',
        Freight: 58.17, Verified: !0
    },
    {
        OrderID: 10254, CustomerID: 'CHOPS', EmployeeID: 5, OrderDate: new
        Date(8370234e5),
        ShipName: 'Chop-suey Chinese', ShipCity: 'Bern', ShipAddress:
        'Hauptstr. 31',
        ShipRegion: 'CJ', ShipPostalCode: '3012', ShipCountry: 'Switzerland',
        Freight: 22.98, Verified: !1
    },
    {
        OrderID: 10255, CustomerID: 'RICSU', EmployeeID: 9, OrderDate: new
        Date(8371098e5),
        ShipName: 'Richter Supermarkt', ShipCity: 'Genève', ShipAddress:
        'Starenweg 5',
        ShipRegion: 'CJ', ShipPostalCode: '1204', ShipCountry: 'Switzerland',
        Freight: 148.33, Verified: !0
    },
    {
        OrderID: 10256, CustomerID: 'WELLI', EmployeeID: 3, OrderDate: new
        Date(837369e6),

```

```

        ShipName: 'Wellington Importadora', ShipCity: 'Resende', ShipAddress:
        'Rua do Mercado, 12',
        ShipRegion: 'SP', ShipPostalCode: '08737-363', ShipCountry: 'Brazil',
        Freight: 13.97, Verified: !1
    },
    {
        OrderID: 10257, CustomerID: 'HILAA', EmployeeID: 4, OrderDate: new
        Date(8374554e5),
        ShipName: 'HILARION-Abastos', ShipCity: 'San Cristóbal', ShipAddress:
        'Carrera 22 con Ave. Carlos Soublette #8-35',
        ShipRegion: 'Táchira', ShipPostalCode: '5022', ShipCountry:
        'Venezuela', Freight: 81.91, Verified: !0
    },
    {
        OrderID: 10258, CustomerID: 'ERNSH', EmployeeID: 1, OrderDate: new
        Date(8375418e5),
        ShipName: 'Ernst Handel', ShipCity: 'Graz', ShipAddress: 'Kirchgasse
        6',
        ShipRegion: 'CJ', ShipPostalCode: '8010', ShipCountry: 'Austria',
        Freight: 140.51, Verified: !0
    },
    {
        OrderID: 10259, CustomerID: 'CENTC', EmployeeID: 4, OrderDate: new
        Date(8376282e5),
        ShipName: 'Centro comercial Moctezuma', ShipCity: 'México D.F.',
        ShipAddress: 'Sierras de Granada 9993',
        ShipRegion: 'CJ', ShipPostalCode: '05022', ShipCountry: 'Mexico',
        Freight: 3.25, Verified: !1
    },
    {
        OrderID: 10260, CustomerID: 'OTTIK', EmployeeID: 4, OrderDate: new
        Date(8377146e5),
        ShipName: 'Ottilies Käseladen', ShipCity: 'Köln', ShipAddress:
        'Mehrheimerstr. 369',
        ShipRegion: 'CJ', ShipPostalCode: '50739', ShipCountry: 'Germany',
        Freight: 55.09, Verified: !0
    },
    {
        OrderID: 10261, CustomerID: 'QUEDE', EmployeeID: 4, OrderDate: new
        Date(8377146e5),
        ShipName: 'Que Delícia', ShipCity: 'Rio de Janeiro', ShipAddress:
        'Rua da Panificadora, 12',
        ShipRegion: 'RJ', ShipPostalCode: '02389-673', ShipCountry: 'Brazil',
        Freight: 3.05, Verified: !1
    },
    {
        OrderID: 10262, CustomerID: 'RATTC', EmployeeID: 8, OrderDate: new
        Date(8379738e5),
        ShipName: 'Rattlesnake Canyon Grocery', ShipCity: 'Albuquerque',
        ShipAddress: '2817 Milton Dr.',
        ShipRegion: 'NM', ShipPostalCode: '87110', ShipCountry: 'USA',
        Freight: 48.29, Verified: !0
    }
}
];

```

INDEX.JSX

```
import { ColumnDirective, ColumnsDirective, Filter, GridComponent, Group,
Inject, Page, Sort } from '@syncfusion/ej2-react-grids';
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { data } from './datasource';
function App() {
    const pageSettings = { pageSize: 6 };
    return <GridComponent dataSource={data} enablePersistence={true}
allowPaging={true} pageSettings={pageSettings}>
        <ColumnsDirective>
            <ColumnDirective field='OrderID' width='100' textAlign="Right"/>
            <ColumnDirective field='CustomerID' width='100' />
            <ColumnDirective field='EmployeeID' width='100'
textAlign="Right"/>
            <ColumnDirective field='Freight' width='100' format="C2"
textAlign="Right"/>
            <ColumnDirective field='ShipCountry' width='100' />
        </ColumnsDirective>
        <Inject services={[Page, Sort, Filter, Group]} />
    </GridComponent>;
}
;
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

INDEX.TSX

```
import { ColumnDirective, ColumnsDirective, Filter, GridComponent, Group,
Inject, Page, PageSettingsModel, Sort } from '@syncfusion/ej2-react-grids';
import * as React from 'react';
import * as ReactDOM from "react-dom";
import { data } from './datasource';
function App() {
    const pageSettings: PageSettingsModel = { pageSize: 6 }
    return <GridComponent dataSource={data} enablePersistence={true}
allowPaging={true} pageSettings={ pageSettings }>
        <ColumnsDirective>
            <ColumnDirective field='OrderID' width='100' textAlign="Right"/>
            <ColumnDirective field='CustomerID' width='100' />
            <ColumnDirective field='EmployeeID' width='100'
textAlign="Right"/>
            <ColumnDirective field='Freight' width='100' format="C2"
textAlign="Right"/>
            <ColumnDirective field='ShipCountry' width='100' />
        </ColumnsDirective>
        <Inject services={[Page, Sort, Filter, Group]} />
    </GridComponent>
};
export default App;
ReactDOM.render(<App />, document.getElementById('element'));
```

State Persistence supported components and properties

The following table illustrates the list of Syncfusion React components that are supported with state persistence and the properties that are stored in the `localStorage`.

<!-- markdownlint-disable MD033 -->

component Name	Properties
Grid	<ul style="list-style-type: none"> Columns filterSettings searchSettings groupSettings pageSettings selectedRowIndex scrollPosition
Accordion	<ul style="list-style-type: none"> expandedIndices
Tabs	<ul style="list-style-type: none"> selectedItem
Schedule	<ul style="list-style-type: none"> currentView selectedDate scrollLeft scrollTop
Kanban	<ul style="list-style-type: none"> columns dataSource swimlaneToggleArray
Chart	<ul style="list-style-type: none"> zoomFactor zoomPosition
Maps	<ul style="list-style-type: none"> zoomSettings
Pivot Table	<ul style="list-style-type: none"> dataSourceSettings pivotValues gridSettings chartSettings displayOption
TreeGrid	<ul style="list-style-type: none"> columns pageSettings searchSettings filterSettings selectedRowIndex sortSettings
Switch, Checkbox	<ul style="list-style-type: none"> checked
RadioButton	<ul style="list-style-type: none"> checked

	<ul style="list-style-type: none"> value
ColorPicker, ListBox, In-placeEditor, RichTextEditor, Autocomplete, Calendar, ComboBox, DatePicker, DropDownList, MaskedTextBox, NumericTextBox, Textbox, TimePicker, Multiselect, DateTimePicker, Slider, Dropdown Tree	<ul style="list-style-type: none"> value
QueryBuilder	<ul style="list-style-type: none"> rule
Splitter	<ul style="list-style-type: none"> paneSettings
DateRangePicker	<ul style="list-style-type: none"> startDate endDate value
Uploader	<ul style="list-style-type: none"> filesData
ListView	<ul style="list-style-type: none"> cssClass enableRtl htmlAttributes enable fields animation headerTitle sortOrder showIcon height width showCheckBox checkBoxPosition
TreeView	<ul style="list-style-type: none"> selectedNodes checkedNodes expandedNodes
Dashboard Layout	<ul style="list-style-type: none"> panels
File Manager	<ul style="list-style-type: none"> view path selectedItems
Sidebar	<ul style="list-style-type: none"> type position isOpen

<!-- markdownlint-enable MD033 -->

Check out the following component's document to learn more about the state persistence.

- [Grid](#)
- [TreeGrid](#)
- [Pivot Table](#)
- [Gantt](#)
- [Kanban](#)
- [Schedule](#)
- [QueryBuilder](#)
- [Tabs](#)

Integration of Material-UI Components

[Material-UI](#) is a popular library for building user interfaces in React applications. Material-UI provides a set of pre-designed and customizable UI components following Material Design principles. These components include buttons, forms, navigation bars, modals, cards, and many others, that can be easily integrated into your application. Material-UI components come with built-in styles and responsive behavior, making it convenient to create visually appealing and consistent user interfaces.

The Syncfusion Material theme is designed to follow the Material-UI guidelines, ensuring a consistent appearance with Material-UI components. This makes it seamless to integrate Syncfusion React components with existing Material-UI components, allowing you to create a unified and visually appealing application.

This guide will provide you with a step-by-step process for integrating Material-UI component into the Syncfusion Grid component.

Set up the React project

1\ Set up a new **React** project by referring to the Syncfusion Grid component's [Getting Started](#) documentation. 2\ Install the necessary **Material-UI** dependencies using the command:

```
`bash
```

```
npm install @mui/material @emotion/react @emotion/styled --save
```

```
,
```

Integrate Material-UI components

Integrate **Material-UI** components with React Grid component effortlessly using the **template** property. Define Material-UI components within the **template** function as shown in the code snippet below:

~/SRC/APP.JSX

```
import * as React from 'react';
import { GridComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-react-grids';
import { Button } from '@mui/material';
import '@syncfusion/ej2-react-grids/styles/material.css';
const dataSource = [
{
```

```
OrderID: 10248, CustomerID: 'VINET'
},
{
OrderID: 10249, CustomerID: 'TOMSP'
},
{
OrderID: 10250, CustomerID: 'HANAR'
}
];
function App() {
  const [data, setData] = React.useState(dataSource);
  function materialTemplate(props) {
    return (
      <div>
        <Button id={props.OrderID} onClick={(e) => {
          setData((prev) => prev.filter((obj) => {
            return obj.OrderID.toString() !== e.target.id;
          }});
        }}>Delete</Button>
      </div>
    );
  }
  return (
    <GridComponent dataSource={data} width='600px' allowKeyboard={false} >
      <ColumnsDirective>
        <ColumnDirective field='OrderID' headerText='Order ID' />
        <ColumnDirective field='CustomerID' headerText='Customer ID' />
        <ColumnDirective template={materialTemplate} width='100px' />
      </ColumnsDirective>
    </GridComponent>
  );
}
```

```
export default App;
```

Run the project

To run the project, use the following command:

```
`bash
```

```
npm start
```

```
`
```

The output will appear as follows:

Order ID	Customer ID	
10248	VINET	DELETE
10249	TOMSP	DELETE
10250	HANAR	DELETE

Integration of Syncfusion React Components in Redux Form

[Redux Form](#) is a popular library used in React project for managing and handling form state. It integrates with the Redux state management library ([Redux Store](#)) and provides a simple and efficient way to handle form inputs, validation, submission, and data synchronization with Redux.

This guide will provide you with a step-by-step process for creating a **Redux Form**, integrating Syncfusion components, and implementing form validation.

Create a Redux Form

To create a simple login form using React Redux Form, follow these steps:

1\ Create a new [React](#) project. 2\ Install the necessary Redux dependencies by executing the following command:

```
`bash
```

```
npm install --save redux react-redux redux-form
```

```
`
```

In this command:

- [Redux](#) is a state container that stores the data.
- [React Redux](#) allows users to read and update the data from the store to the React components.
- [Redux Form](#) is used to manage the form state in React. 3\ Create an empty form by creating a new file called **LoginForm.js** and defining the form component:

LOGINFORM.JS

```
import { reduxForm } from "redux-form";  
let LoginForm = (props) => {  
  return (  
    <form />  
  );  
};
```

4\ Include the form component in your React project by adding the following code to the **App.js** file:

APP.JS

```
import './App.css';  
import LoginForm from './LoginForm';  
function App() {  
  return (  
    <div className="App">  
      <div className="login-form">  
        <h2>Redux form</h2>  
        <LoginForm />  
      </div>  
    </div>  
  );  
}  
export default App;
```

Setting up the Redux store

After creating the form and integrating it into the React project, set up a store in the **index.js** file to store the form state using the **Provider** component. This allows the **Redux store** to be accessible to the child components.

INDEX.JS

```
import { Provider } from 'react-redux';
import { createStore, combineReducers } from 'redux';
import { reducer as formReducer } from 'redux-form';
const appReducer = combineReducers({ form: formReducer });
const store = createStore(appReducer);
ReactDOM.render(<Provider store={store}><App /></Provider>,
document.getElementById('root'));
```

Connect with Redux Form

In the **LoginForm.js** file, connect the form component with **Redux Form** by exporting it using the **reduxForm** higher-order component:

LOGINFORM.JS

```
export default LoginForm = reduxForm({
  form: "login"
})(LoginForm);
```

Add Syncfusion React components

To incorporate Syncfusion React components (DatePicker, TextBox, Button) into the login form, follow these steps:

1\ Install the necessary dependencies for React components.

```
`bash
```

```
npm install @syncfusion/ej2-react-calendars @syncfusion/ej2-react-inputs @syncfusion/ej2-react-buttons --save
```

```
`
```

2\ Integrate the React components within the login form using the **Field** component provided by Redux Form.

LOGINFORM.JS

```
import { reduxForm, Field } from "redux-form";
import { DatePickerComponent } from '@syncfusion/ej2-react-calendars';
import { TextBoxComponent } from '@syncfusion/ej2-react-inputs';
import { ButtonComponent } from '@syncfusion/ej2-react-buttons';
const textBox = ({ placeholder }) => {
  return <TextBoxComponent placeholder={placeholder} floatLabelType="Auto" />
};
const datePicker = ({ placeholder }) => {
  return <DatePickerComponent placeholder={placeholder} floatLabelType="Auto" />
};
let LoginForm = () => {
  return (
```

```

<form>
<Field name="username" component={textBox} placeholder="Enter the user name"
/>
<Field name="password" component={textBox} placeholder="Enter the password"
/>
<Field name="dob" component={datePicker} placeholder="Enter the date of
birth" />
<ButtonComponent type="submit">Submit</ButtonComponent>
</form>
);
};
export default LoginForm = reduxForm({
  form: "login"
})(LoginForm);

```

3\ Define a **handler** function to handle form submission and display the form data in the **App.js** file:

APP.JS

```

function App() {
  const handleLogin = values => {
    console.log(`User name: ${values.username}`);
    console.log(`Password: ${values.password}`);
    console.log(`DOB: ${values.dob}`);
  }
  return (
    <div className="App">
      <div className="login-form">
        <h2>Redux form</h2>
        <LoginForm onSubmit={handleLogin} />
      </div>
    </div>
  );
}
export default App;

```

LOGINFORM.JS

```

let LoginForm = (props) => {
  const { handleSubmit } = props;
  return (
    <form onSubmit={handleSubmit}>
      ...
    </form>
  );
};

```

Form Validation

Redux Form provides built-in functionality for form validation. Implement the **validate** function to validate the form and display error messages when needed. The form inputs will be automatically validated upon submission.

APP.JS

```

const validate = values => {

```

```
const errors = {};  
if (!values.username) {  
  errors.username = 'Required';  
}  
if (!values.password) {  
  errors.password = 'Required';  
}  
if (!values.dob) {  
  errors.dob = 'Required';  
}  
return errors;  
}  
export default LoginForm = reduxForm({  
  form: "login",  
  validate  
})(LoginForm);
```

Displaying Errors

Upon form submission, the inputs will be validated and corresponding error messages will be displayed below the respective input fields.

APP.JS

```
const textBox = ({ placeholder, input, meta: { touched, error } }) => {  
  return <div>  
    <TextComponent placeholder={placeholder} floatLabelType="Auto" {...input}  
    onChange={ (e) => {  
      input.onChange(e.target.value);  
    }}  
    />  
    {touched && error && <span className="error">{error}</span>}  
  </div>  
};  
const datePicker = ({ placeholder, input, meta: { touched, error } }) => {  
  return <div>  
    <DatePickerComponent placeholder={placeholder} floatLabelType="Auto"  
    {...input}  
    onChange={ (e) => {  
      input.onChange(e.target.value);  
    }}  
    />  
    {touched && error && <span className="error">{error}</span>}</div>  
};
```

Run the project

To run the project, use the following command:

```
`bash
```

```
npm start
```

```
`
```

Refer to the following output image.

Redux form

Enter the user name

Required

Enter the password

Required

Enter the date of birth



Required

SUBMIT

Refer to the [GitHub repository](#) for a runnable demo.

How To

How to resolve Content Security Policy (CSP) errors

Enabling the strict Content Security Policy (CSP) may cause the following issue with the Syncfusion React components in your application.

Image loading

Syncfusion license banner utilize the image from **base64**, which is not allowed on strict CSP-enabled sites. To overcome this restriction, it is necessary to add the [img-src data:](#) directive in the meta tag or consider [registering the license key](#).

TroubleShoot

Content Security Policy

Content Security Policy (CSP) is a security feature implemented by web browsers that helps to protect against attacks such as cross-site scripting (XSS) and data injection. It limits the sources from which content can be loaded on a web page.

To enable strict [Content Security Policy \(CSP\)](#), certain browser features are disabled by default. In order to use Syncfusion React components with strict CSP mode, it is essential to include following directives in the CSP meta tag.

- Syncfusion components are rendered with calculated **inline styles** and **base64** font icons, which are blocked on a strict CSP-enabled site. To allow them, add the [style-src 'self' 'unsafe-inline';](#) and [font-src 'self' data:;](#) directives in the meta tag as follows.

HTML

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self';
style-src 'self' 'unsafe-inline';
font-src 'self' data:;" />
```

- Syncfusion **material** and **tailwind** built-in themes contain a reference to the [Roboto's external font](#), which is also blocked. To allow them, add the [external font](#) reference to the [style-src](#) and [font-src](#) directives in the above meta tag.

The resultant meta tag is included within the `<head>` tag and resolves the CSP violation on the application's side when utilizing Syncfusion React components with material and tailwind themes.

HTML

```
<head>
...
<meta http-equiv="Content-Security-Policy" content="default-src 'self';
style-src 'self' https://fonts.googleapis.com/ 'unsafe-inline';
font-src 'self' https://fonts.googleapis.com/ https://fonts.gstatic.com/
data:;" />
</head>
```

Note: From the 2023 Vol2 - 22.1 release onwards, the Content Security Policy for Syncfusion React components has been enhanced. The usage of the `unsafe-eval` directive has been eliminated from the CSP meta tag.

[View the React sample enabled with strict CSP in Github](#)

See also

- [How to resolve the Content Security Policy \(CSP\) errors](#)

Visual Studio Code Integration

Visual Studio Code Integration

Overview

The Syncfusion Essential Studio Web extension for Visual Studio Code allows you to use the Syncfusion JavaScript components(React, Angular, and Vue) easily by configuring the Syncfusion NPM packages and themes.

The Syncfusion Web Extension provides the following support in Visual Studio Code:

[Project-Template](#): Creates Syncfusion Web applications by adding the required Syncfusion JavaScript components.

Download and Installation

Syncfusion publishes the Visual Studio Code extension in [Visual Studio Code marketplace](#). You can either install it from Visual Studio Code or download and install it from the Visual Studio Code marketplace.

Prerequisites

The following prerequisites software needs to be installed for the Syncfusion Web extension installation and for creating the Syncfusion Web applications along with any one of the Framework(React, Angular, and Vue).

- [Visual Studio Code](#)

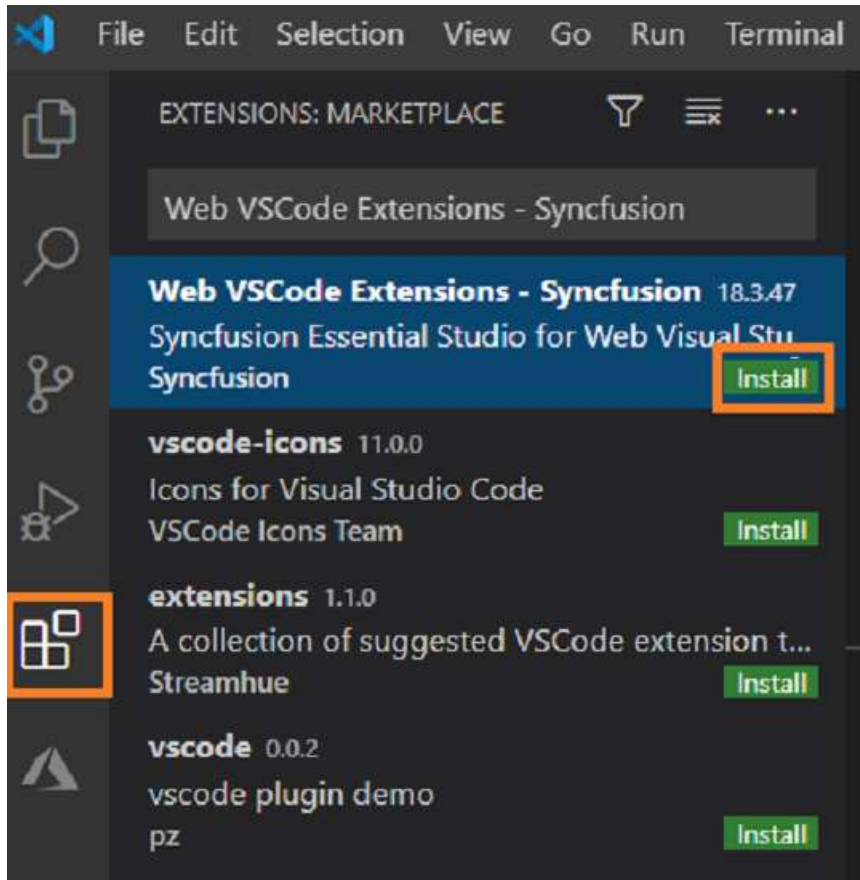
The minimum version of the Visual Studio Code is 1.38.0 to use the Syncfusion Web Extension.

- [C# Extension](#)
- [Node.js](#)

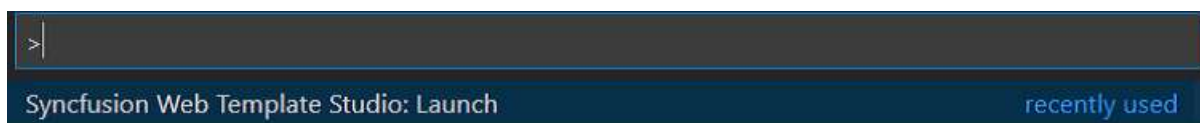
Install through the Visual Studio Code extensions

The following steps explain how to install the Syncfusion Web extensions from Visual Studio Code Extensions.

1. Open Visual Studio Code.
2. Go to **View > Extensions**, and open Manage Extensions.
3. Type “**Syncfusion Web**” in the search box.



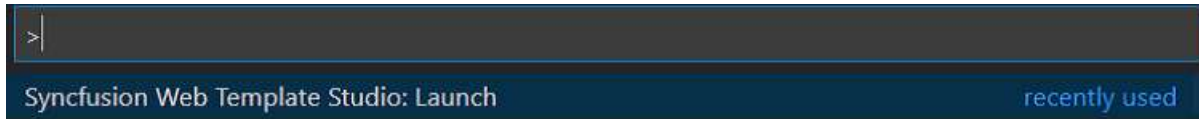
4. Click the Install button on the “**Web VSCode Extensions - Syncfusion**” extension.
5. After the installation, reload the Visual Studio Code using the **Reload Required** in Visual Studio Code palette.
6. Now, use the Syncfusion Web extensions from the Visual Studio Code palette.



Install from the Visual Studio Code Marketplace

The following steps explain how to download Syncfusion Web applications from the Visual Studio Code Marketplace and install them.

1. Open the [Syncfusion Web Extension](#)
2. Click Install from Visual Studio Code Marketplace. The browser opens the popup with the information like “**Open Visual Studio Code**”. Click Open Visual Studio Code, then [Syncfusion Web Extension](#) will open in Visual Studio Code.
3. Click the Install button in the “**Web VSCode Extensions - Syncfusion**” extension.
4. After the installation, reload the Visual Studio Code using the Reload Required in Visual Studio Code palette.
5. Now, use the Syncfusion Web extensions from the Visual Studio Code palette.



Visual Studio Code Extensions

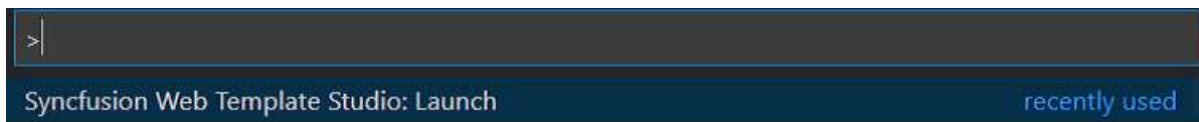
Create project

Synconfusion provides **project templates** for **Visual Studio Code** to create Synconfusion Web applications. The Synconfusion Web Project template creates applications with the selected Framework (React, Angular, and Vue), required Synconfusion NPM packages, component render code for the Grid, Chart, and Scheduler components, and a style to make development with Synconfusion components easier.

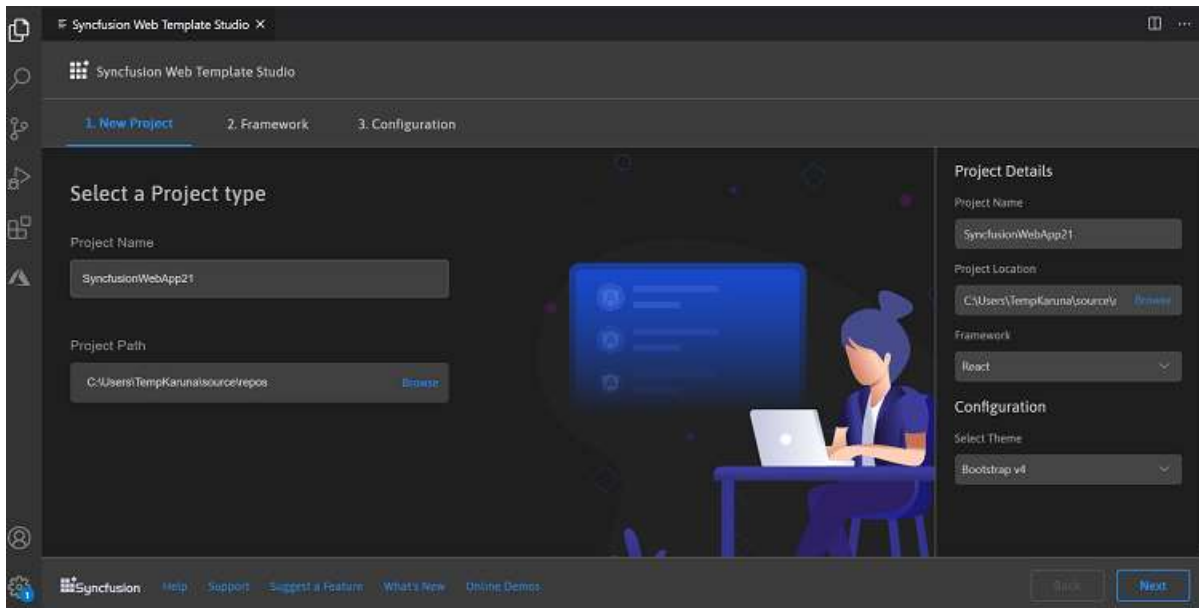
The Synconfusion Visual Studio Code project template provides support for Web project templates from v18.3.0.47.

The steps below help you to create **Synconfusion Web Applications** through the **Visual Studio Code**:

1. In Visual Studio Code, open the command palette by pressing **Ctrl+Shift+P**. The Visual Studio Code palette opens, search the word **Synconfusion**, so you can get the templates provided.

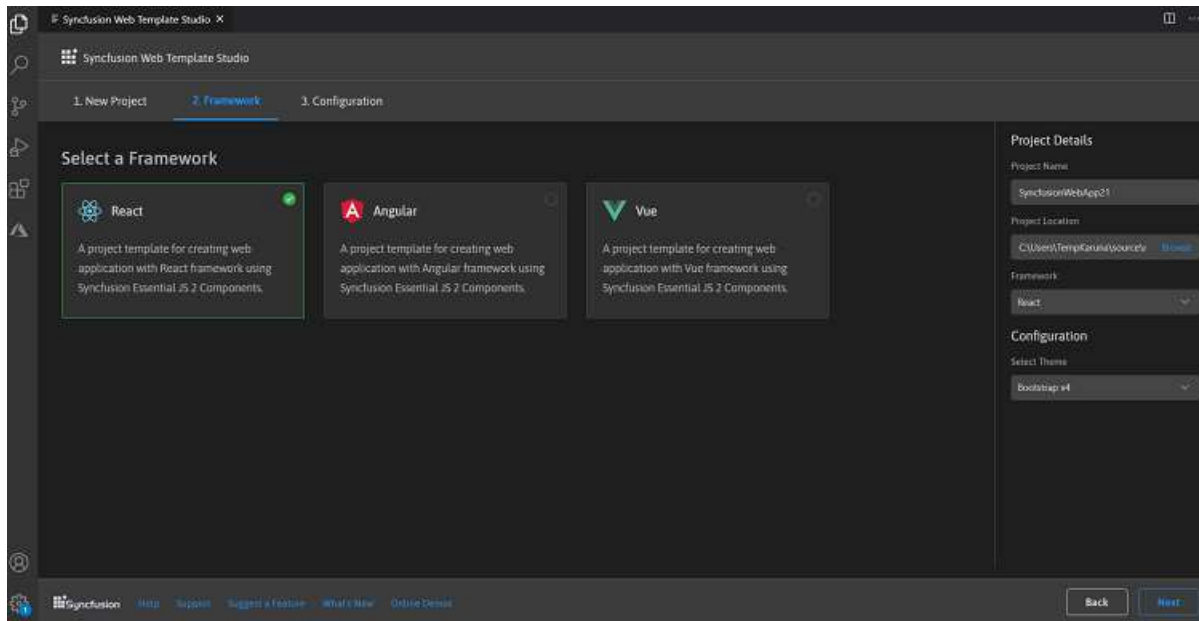


2. Select **Synconfusion Web Template Studio: Launch** and then press enter, Template Studio wizard for configuring the Synconfusion Web app will appear. Provide the require Project Name and Path to create the new Synconfusion Web application along with any one of the Framework (React, Angular, and Vue).

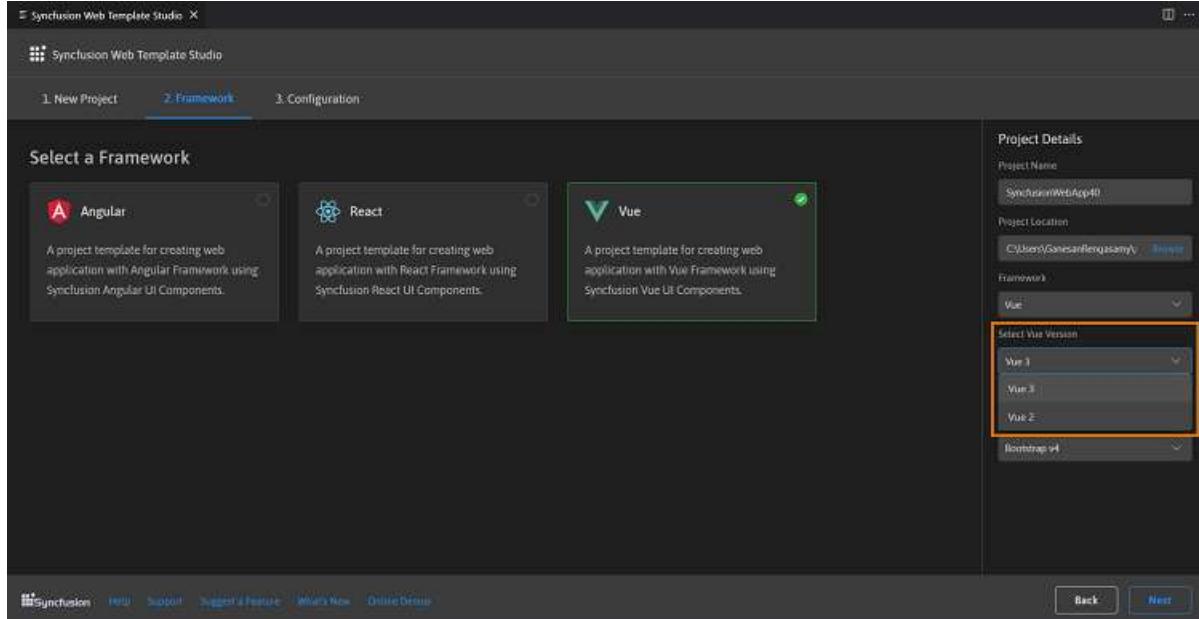


3. Click either **Next** or **Framework** tab, and the Framework types will be appears. Choose any one of the Framework:

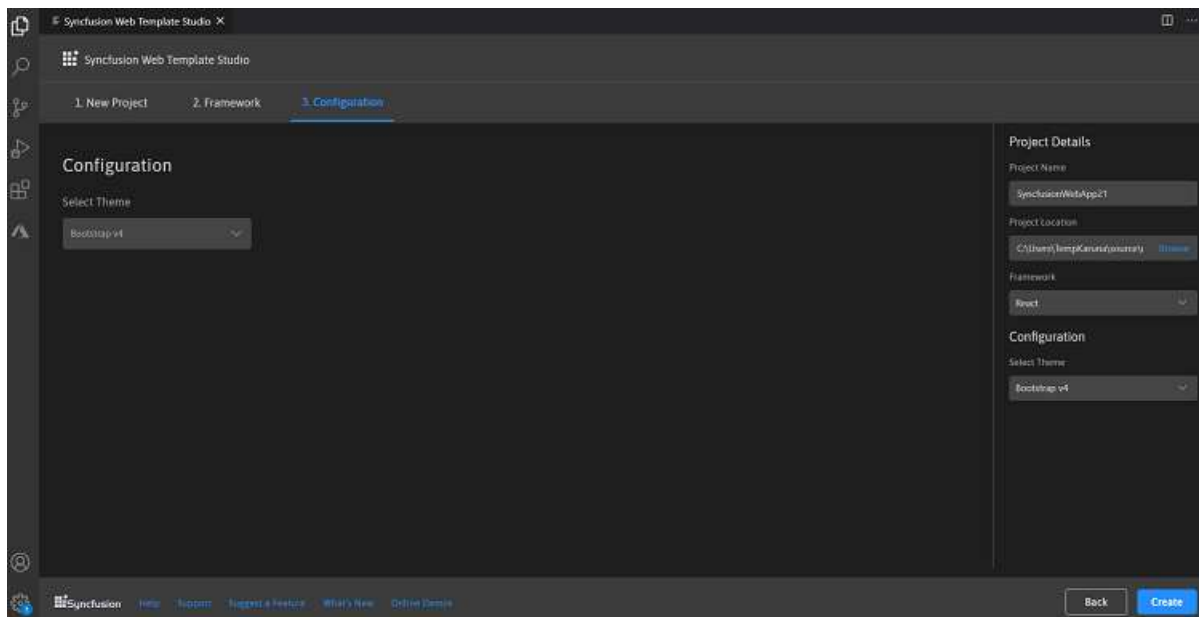
- React
- Angular
- Vue



If you choose the Vue framework, the **Select Vue Version** option will appear in the **Project Details** section. You can create the Vue application using either the Vue 3 or Vue 2 versions.



4. Click either **Next** or the **Configuration** tab, and the Configuration section will be loaded. Choose the preferred theme and then click **Create**. The project will be created.



5. The created Syncfusion Web App is configured with the Syncfusion NPM packages, styles, and the component render code for the Syncfusion component added.

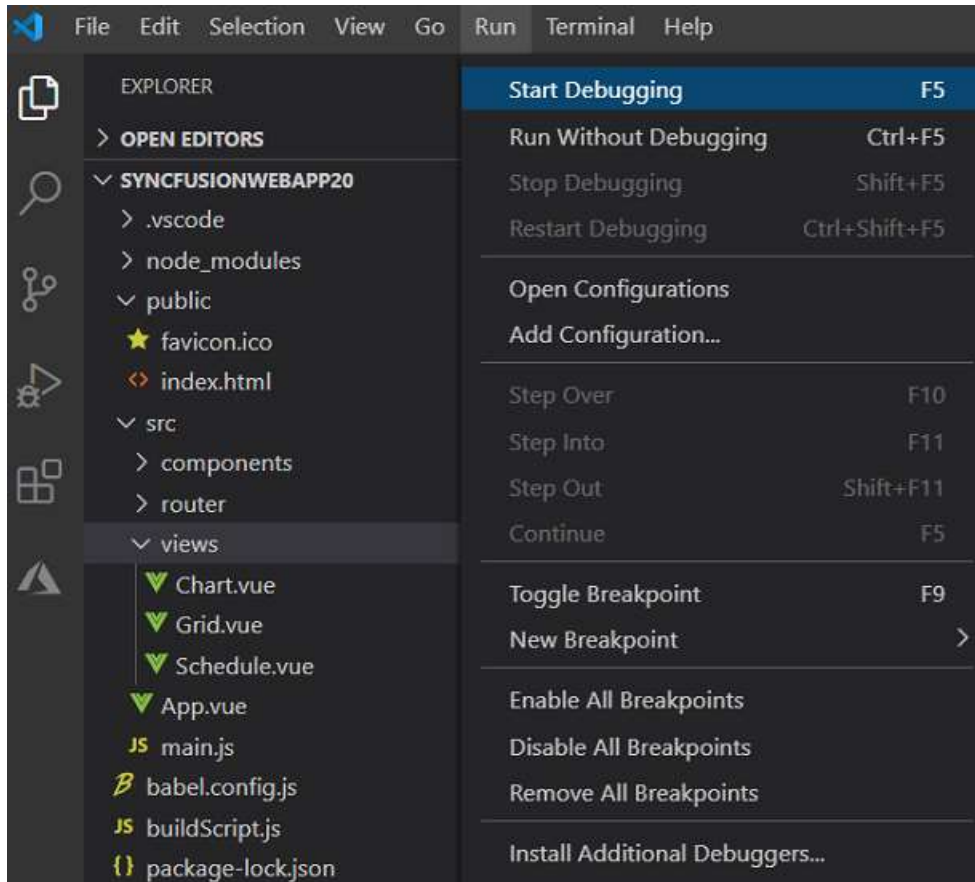
```
0 package.json > ...
1 {
2   "name": "SyncfusionWebApp21",
3   "version": "0.1.0",
4   "private": true,
5   "dependencies": {
6     "@syncfusion/ej2-react-charts": "^18.3.46",
7     "@syncfusion/ej2-react-grids": "^18.3.46",
8     "@syncfusion/ej2-react-schedule": "^18.3.46",
9     "bootstrap": "4.5.0",
10    "classnames": "2.2.6",
11    "cookie-parser": "1.4.5",
12    "debug": "4.1.1",
13    "express": "4.17.1",
14    "fs-extra": "9.0.1",
15    "http-errors": "1.8.0",
16    "morgan": "1.10.0",
17    "react": "16.13.1",
18    "react-dom": "16.13.1",
19    "react-router-dom": "5.2.0",
20    "react-scripts": "3.4.1",
21    "uuid": "8.2.0"
22  },
23  "eslintConfig": {
24    "extends": "react-app"
25  },
26  "devDependencies": {
27    "concurrently": "5.2.0"
28  }
29 }
```

```
src > # App.css > ...
1  @import './node_modules/@syncfusion/ej2-base/styles/bootstrap4.css';
2  @import './node_modules/@syncfusion/ej2-buttons/styles/bootstrap4.css';
3  @import './node_modules/@syncfusion/ej2-calendars/styles/bootstrap4.css';
4  @import './node_modules/@syncfusion/ej2-dropdowns/styles/bootstrap4.css';
5  @import './node_modules/@syncfusion/ej2-inputs/styles/bootstrap4.css';
6  @import './node_modules/@syncfusion/ej2-navigations/styles/bootstrap4.css';
7  @import './node_modules/@syncfusion/ej2-popups/styles/bootstrap4.css';
8  @import './node_modules/@syncfusion/ej2-splitbuttons/styles/bootstrap4.css';
9  @import './node_modules/@syncfusion/ej2-react-grids/styles/bootstrap4.css';
10 @import './node_modules/@syncfusion/ej2-react-schedule/styles/bootstrap4.css';
11
```

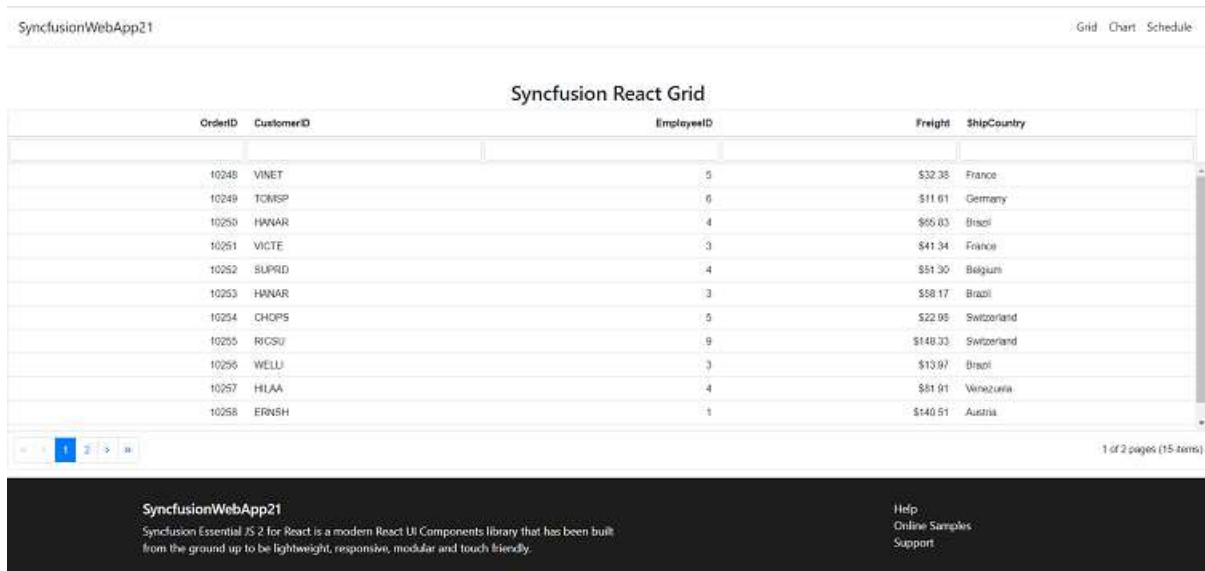


Run the application

1. Click on **F5** or navigate to **Run>Start debugging**



2. After compilation process completed, open the local host link in browser to see the output.



Add SynCFusion component to the application

We have showcased the Chart, Grid and Schedule component in SynCFusion web application. If you want to create your application with other SynCFusion components, you need to install the required

component package and then you can add it in your application. To know about npm package installation, refer to the [installation](#) section.

Upgrading the npm packages

While creating the new Syncfusion web app, it install the npm packages with latest version. If you want to use your existing project in future, you can update the npm packages without uninstalling it. Refer to the [update npm packages](#) section for upgrading the package.

Visual Studio Integration

Visual Studio Integration

Overview

The Syncfusion Essential Studio for React extensions for Visual Studio that allow you to use the Syncfusion component easier by creating Syncfusion React application in Visual Studio and option upgrading the Syncfusion version from the application.

The Syncfusion React provides the following supports in Visual Studio:

1. [Project template](#): Creates the Syncfusion React application by adding the required Syncfusion React components.
2. [Convert project](#): Converts an existing React application into a Syncfusion React application by adding the required Syncfusion NPM and resource files.
3. [Migrate project](#): Upgrades the existing Syncfusion React application from one Essential Studio version to another version.

In Visual Studio 2017, you can see the Syncfusion menu directly in the Visual Studio menu.

Visual Studio Integration

Create project

Syncfusion provides the **Visual Studio Project Templates** for creating the Syncfusion React Application. The Syncfusion React application creates the application with the required Syncfusion references, namespaces and CDN links for making the development earlier with the Syncfusion components.

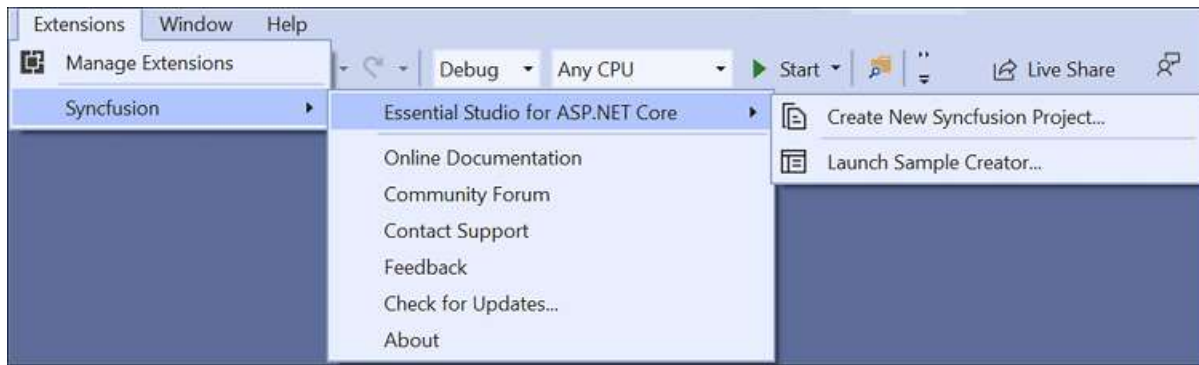
The Syncfusion React project templates are available from v17.1.0.47.

The following steps help you to create the Syncfusion React Application through the Visual Studio:

1. Open the Visual Studio 2017 or later.
2. To create a Syncfusion React project, follow either one of the options below:

Option 1:

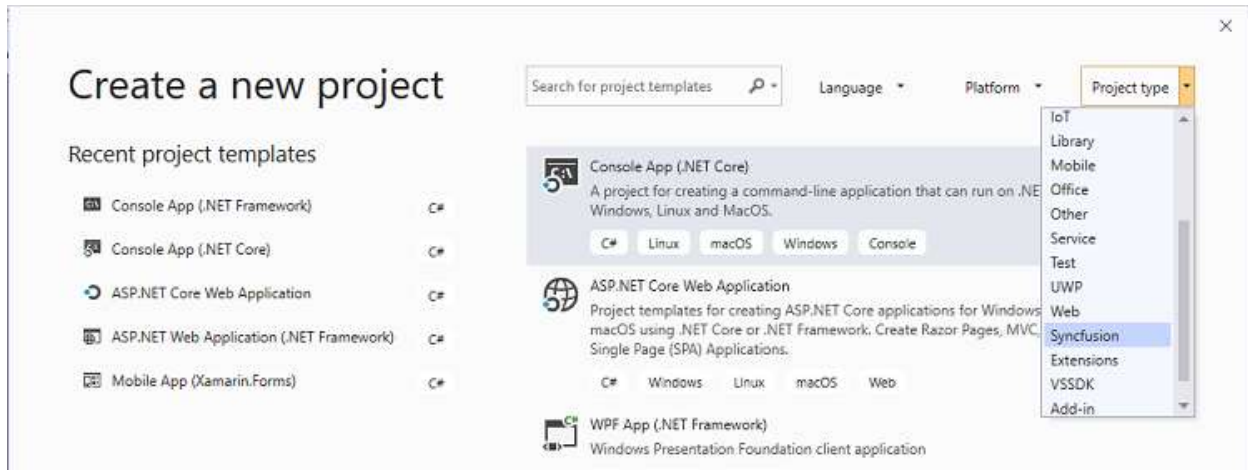
Choose the **Extension->Syncfusion-> Essential Studio for ASP.NET Core -> Create New Syncfusion Project...** in the Visual Studio menu.



In Visual Studio 2017, you can see the **Syncfusion** menu directly in the Visual Studio menu.

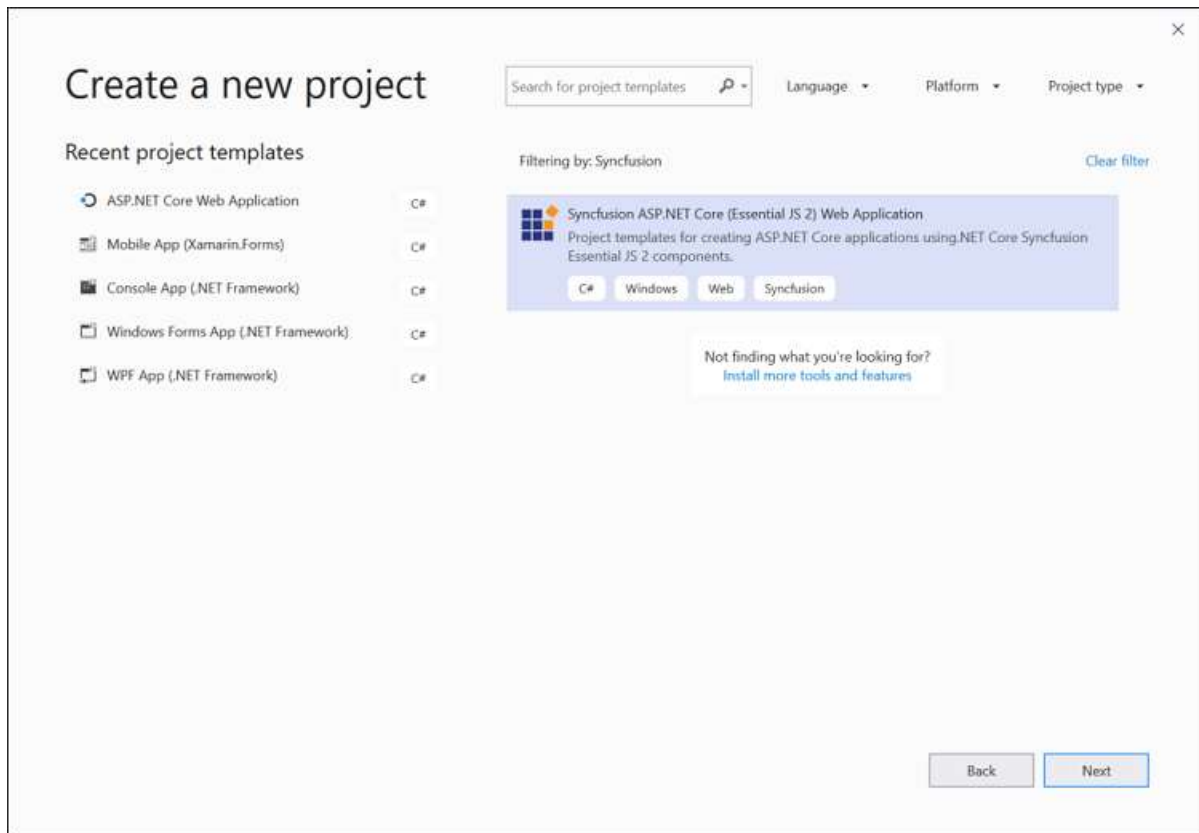
Option 2:

Choose **File > New > Project** in Visual Studio. The Create a new project dialog opens. You can get the Syncfusion provided templates by filtering the Project type with Syncfusion or use the **Search option** with the key word of **Syncfusion**.

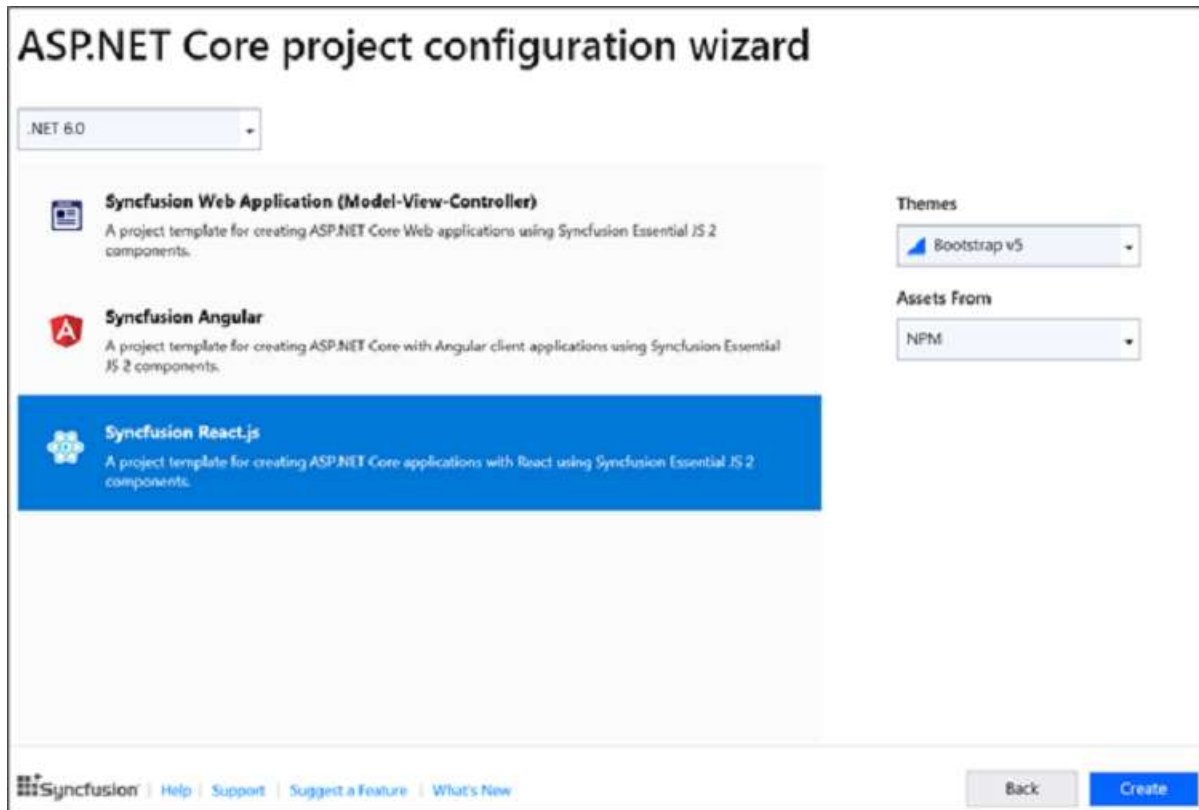


In Visual Studio 2017, choose **File > New > Project** and navigate to **Syncfusion > .NET Core > Syncfusion ASP.NET Core (Essential JS 2) Web Application** in Visual Studio.

3. Select the **Syncfusion ASP.NET Core Web Application** and choose the Next button.



4. Name the **Project**, choose the destination location and then click **Create** button. The **Syncfusion ASP.NET Core** project configuration wizard appears.



Choose the **Synfusion React.js** template and choose required theme and asset.

5. Click the Create button, the Synfusion React application has been created.
6. The created Synfusion React application configured with Synfusion.
7. The required Synfusion React NPM packages, scripts and selected style configured with the application.

Convert Project

Synfusion React conversion is a Visual Studio add-in that converts an existing React application into a Synfusion React Web application.

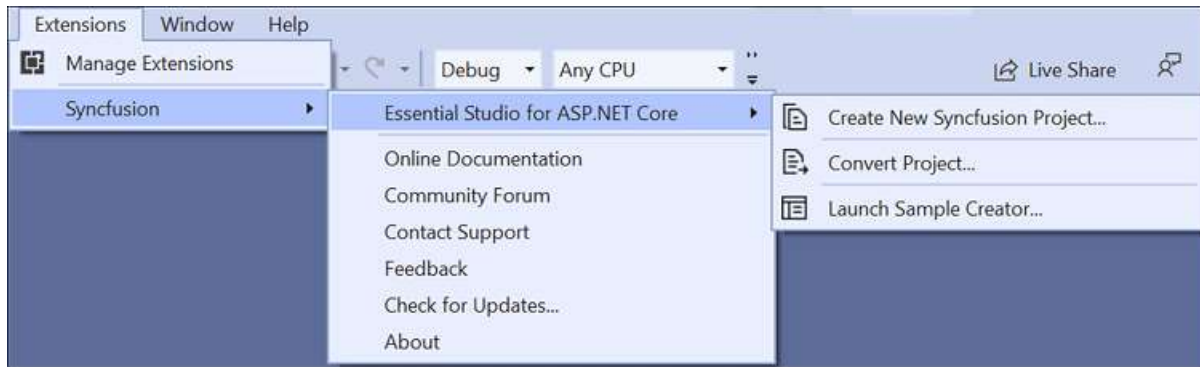
The Synfusion React Project conversion are available from v17.3.0.9.

The steps below help you to convert the React application to Synfusion React application through the Visual Studio:

1. Open your existing React application or create a new React application
2. To open the Synfusion Project Conversion Wizard, follow either one of the options below:

Option 1:

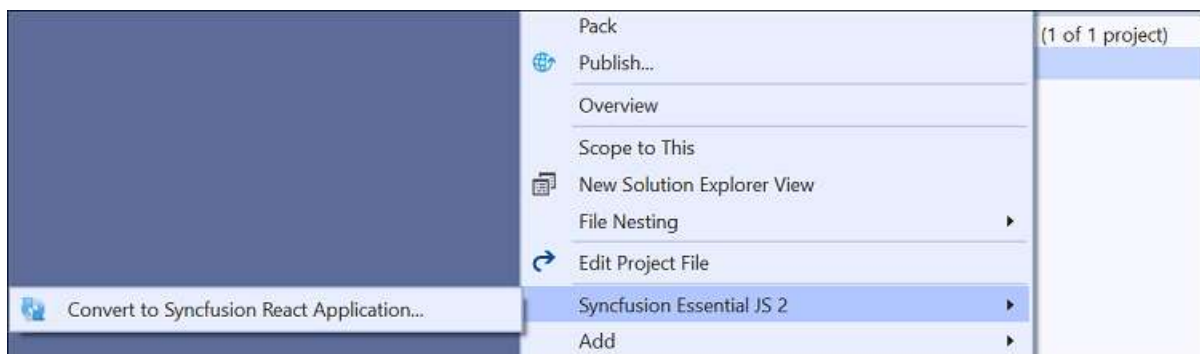
Choose **Extensions-> Synfusion-> Essential Studio for ASP.NET Core ->Convert Project...** in **Visual Studio** menu.



In Visual Studio 2017, you can see the **Syncfusion** menu directly in the Visual Studio menu

Option 2:

Right-click on the **React Application** from the Solution Explorer and select the **Syncfusion Web** and choose the **Convert to Syncfusion React application...**



3. The **Syncfusion React Project Conversion** window will appear. You can choose the required version of Syncfusion React version, Assets from, and Themes to convert the application.

Project Conversion - React

Version : 19.4.0.41

Configurations

Select Version

19.4.38

Select Asset From

NPM

Select Theme

Bootstrap v5

☐ Enable a backup before converting

C:\Users\AbishakeDakshinamoor\source\repos\SyncfusionReactProje

Syncfusion

[Help](#) [Support](#) [Suggest a Feature](#) [What's New](#) [Online Demos](#)

Cancel Convert

The Syncfusion React versions are loaded from published Syncfusion React NPM package versions and it requires the internet connectivity.

The following configurations are used in the Project conversion wizard.

Assets From: Load the Syncfusion Essential JS 2 assets to React Project, from either NPM, CDN, or Installed Location.

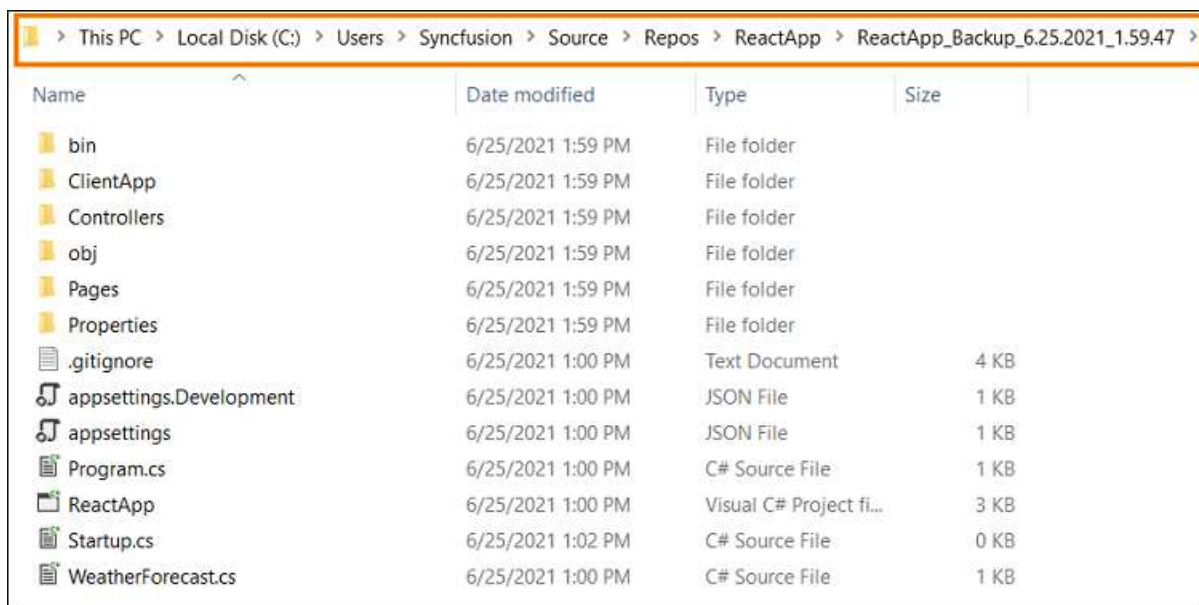
Installed location option will be available only when the Syncfusion Essential JavaScript 2 setup has been installed.

Choose the Theme: Choose the required theme.

4. Check the **“Enable a backup before converting”** checkbox if you want to take the project backup and choose the location.
5. Once the conversion process completed, will get the success message window.



if you enabled project backup before converting, the old project was saved in the specified backup path location, as shown below once the conversion process completed.



6. The required Syncfusion React NPM packages with selected version, scripts and selected style are added in the application.

Upgrade Project

The Syncfusion React migration add-in for Visual Studio allows you to migrate an existing Syncfusion React application from one version of Essential Studio version to another version. This reduces the amount of manual work required when migrating the Syncfusion version.

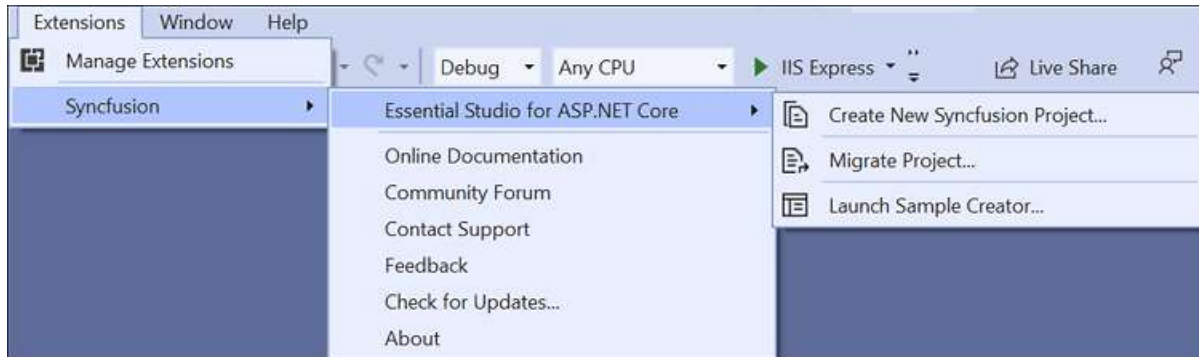
The Syncfusion React Project migration are available from v17.3.0.9.

The steps below help you to upgrade the Syncfusion version in **Syncfusion React Application** through the **Visual Studio**:

1. Open the Syncfusion React application which uses the Syncfusion component.
2. To open Migration Wizard, follow either one of the options below:

Option 1:

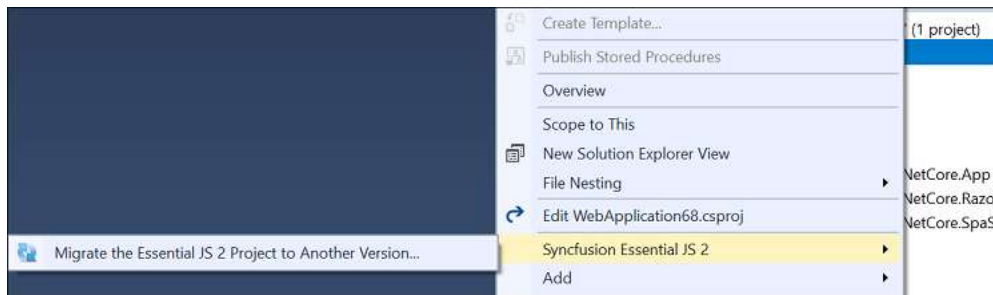
Choose **Extensions-> Syncfusion-> Essential Studio for ASP.NET Core ->Migrate Project...** in **Visual Studio** menu.



In Visual Studio 2017, you can see the **Syncfusion** menu directly in the Visual Studio menu

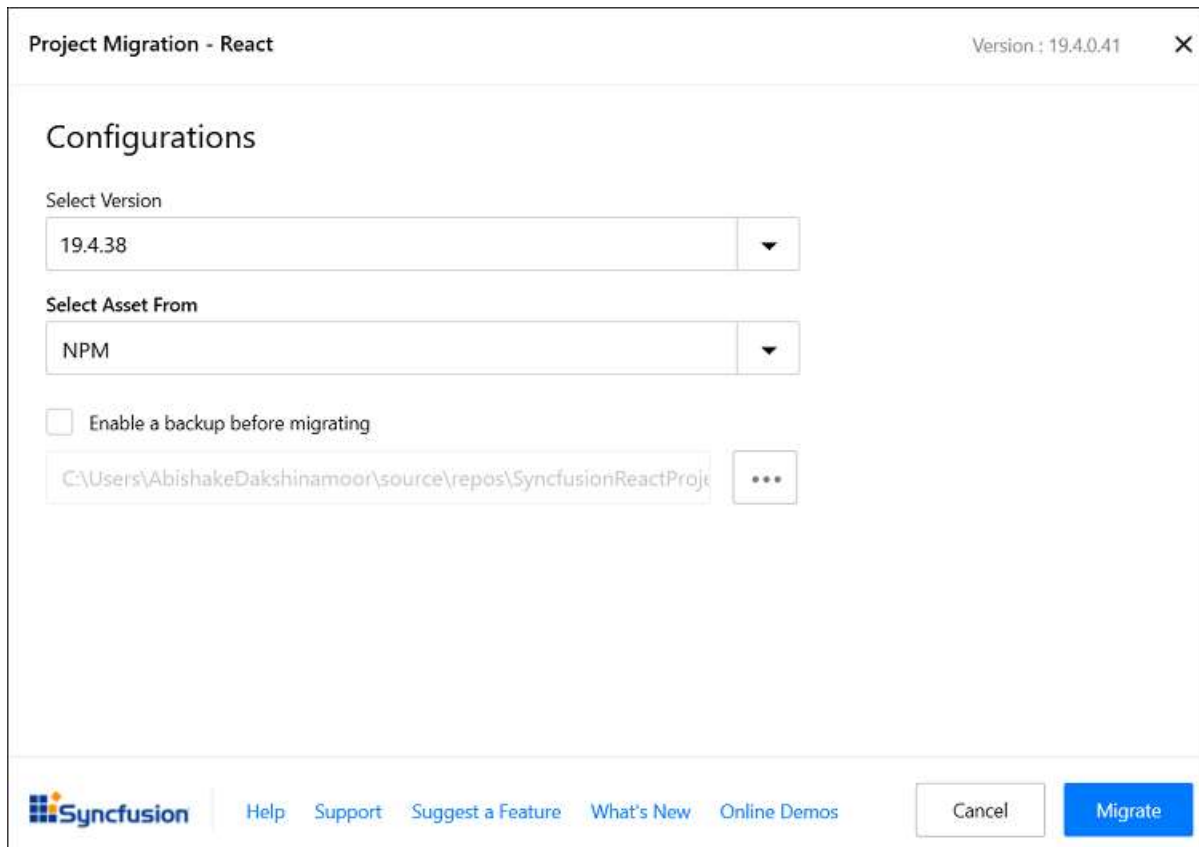
Option 2:

Right-click on the **Application** from the **Solution Explorer** and select the **Syncfusion Web** and choose the **Migrate the Syncfusion ASP.NET Core Project to Another version...**



3. The Syncfusion Project Migration window will appear. You can choose the required version of Syncfusion React to migrate.

The Syncfusion React versions are loaded from published Syncfusion React NPM packages and it requires the internet connectivity.



Project Migration - React Version : 19.4.0.41


Configurations

Select Version
19.4.38

Select Asset From
NPM

☐ Enable a backup before migrating

C:\Users\AbishakeDakshinamoor\source\repos\SyncfusionReactProje ...

 [Help](#) [Support](#) [Suggest a Feature](#) [What's New](#) [Online Demos](#) Cancel Migrate

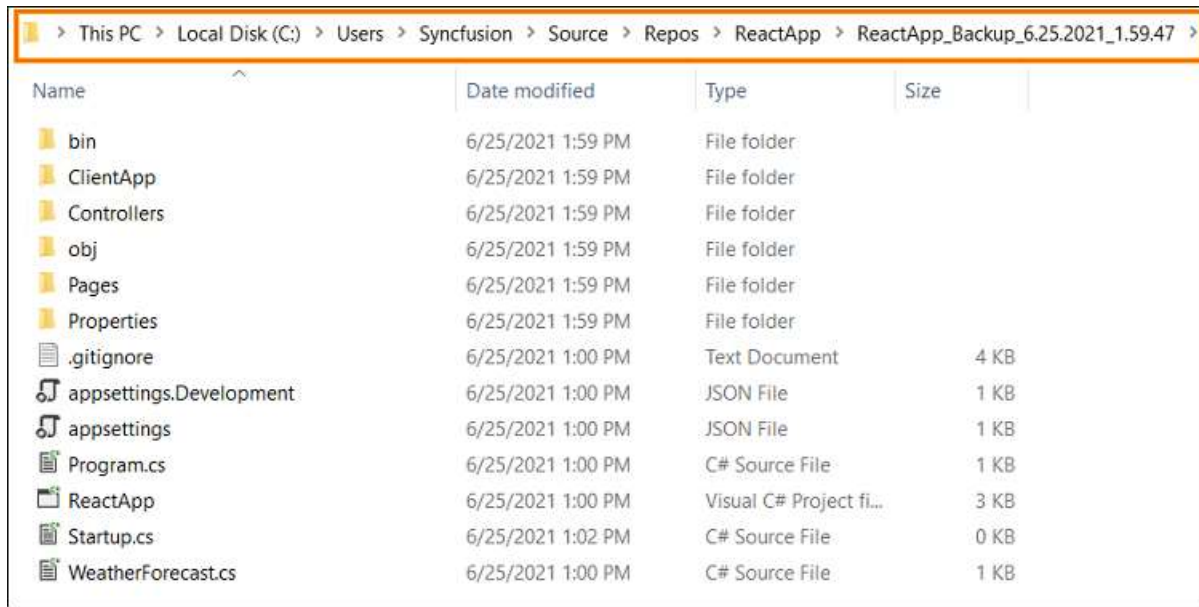
Assets From: Load the Syncfusion Essential JS 2 assets to React Project, from either NPM, CDN, or Installed Location.

Installed location option will be available only when the Syncfusion Essential JavaScript 2 setup has been installed.

4. Check the **“Enable a backup before migrating”** checkbox if you want to take the project backup and choose the location.
5. Once the migration process completed, will get the success message window.



if you enabled project backup before migrating, the old project was saved in the specified backup path location, as shown below once the migration process completed



Name	Date modified	Type	Size
bin	6/25/2021 1:59 PM	File folder	
ClientApp	6/25/2021 1:59 PM	File folder	
Controllers	6/25/2021 1:59 PM	File folder	
obj	6/25/2021 1:59 PM	File folder	
Pages	6/25/2021 1:59 PM	File folder	
Properties	6/25/2021 1:59 PM	File folder	
.gitignore	6/25/2021 1:00 PM	Text Document	4 KB
appsettings.Development	6/25/2021 1:00 PM	JSON File	1 KB
appsettings	6/25/2021 1:00 PM	JSON File	1 KB
Program.cs	6/25/2021 1:00 PM	C# Source File	1 KB
ReactApp	6/25/2021 1:00 PM	Visual C# Project fi...	3 KB
Startup.cs	6/25/2021 1:02 PM	C# Source File	0 KB
WeatherForecast.cs	6/25/2021 1:00 PM	C# Source File	1 KB

6. The Syncfusion React NPM packages, and CSS are updated to the selected version in the project.