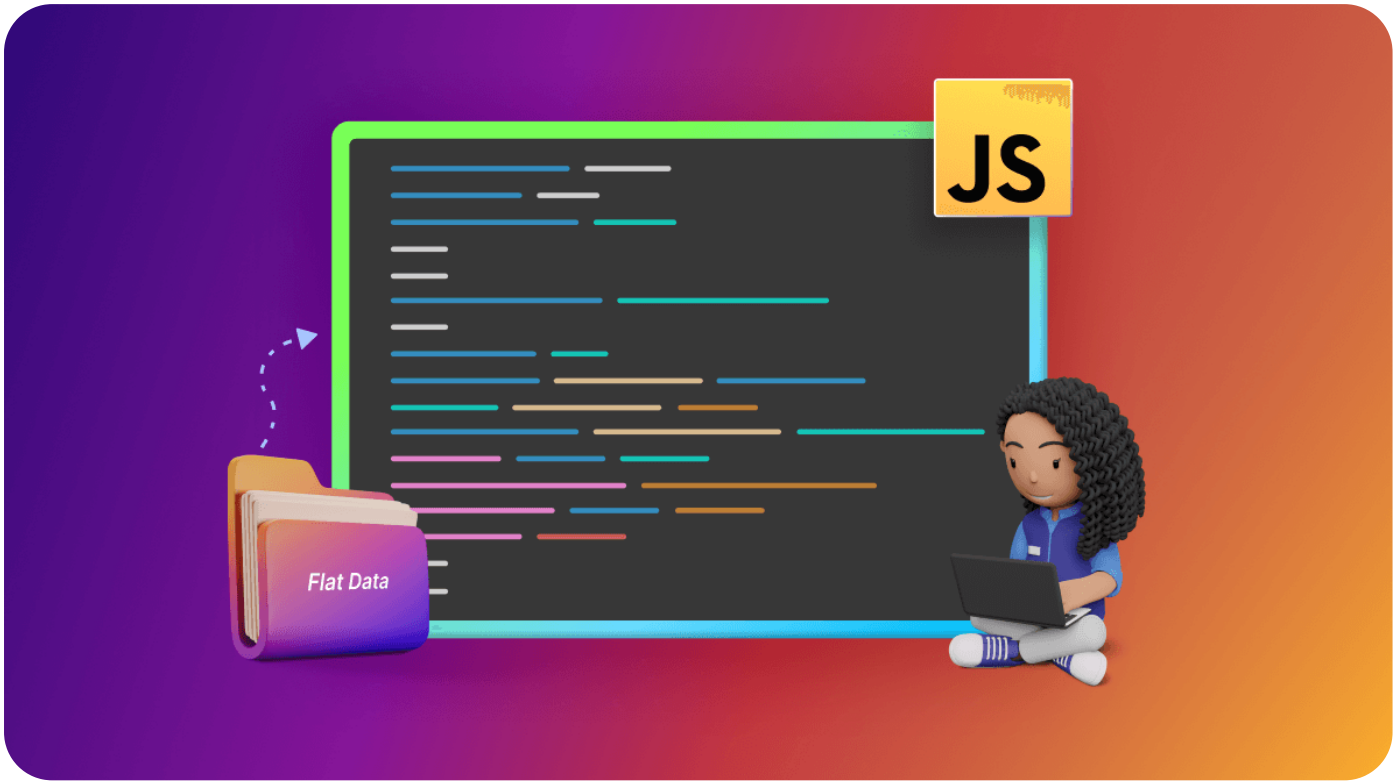
Easily Render Flat JSON Data in JavaScript File Manager

Keerthana Rajendran • 11 min read • Jan 8, 2025 • **Updated**



> **TL;DR:** *Explore the power of flat JSON data rendering in #JavaScript File Manager. This feature allows you to manage your file system and perform common file operations without the need for HTTP client requests and backend URL configuration. Learn how to leverage this feature with your required services and enhance your file management experience.*

The Syncfusion JavaScript File Manager component is a graphical user interface that manages the file system. This component provides easy navigation for browsing and selecting files and folders from the file system. You can perform the most common file and folder operations, such as read, write, delete, create, rename, upload, edit, select, and sort.

From the 2024 Volume 2 onward, the File Manager supports rendering an array of flat JSON data.

Let’s explore this new feature in detail!

What is flat data?

Flat JSON data is a collection of files and folders stored with a defined structure. These objects can also be defined locally or retrieved from any file service provider through File Manager events.

Why flat data?

The Syncfusion JavaScript File Manager can be populated with flat JSON data, eliminating the need for HTTP client requests and backend URL configuration. This allows you to utilize your required services, such as physical, Amazon, Azure, etc., through the File Manager's action events. This supports all file operations like delete, cut, copy, paste, new folder creation, upload, download, and more.

Cloud service providers such as Google Drive have facilitated the option to fetch the data using client-side JavaScript; refer to the documentation for more details. So, the JavaScript File Manager now enables rendering the component with complete JavaScript code without maintaining a separate service provider.

Render flat JSON data in JavaScript File Manager

Let’s see how to render flat JSON data in the JavaScript File Manager component by following these steps!

Step 1: Setting up the TypeScript app

First, create a TypeScript app. This can be done by referring to the instructions in the getting started with JavaScript File Manager documentation.

Step 2: Initialize the JavaScript File Manager

Now, initialize the JavaScript File Manager in the app using the following code.

```
<div class="sample-container">
 <!-- Initialize FileManager -->
 <div id="filemanager"></div>
</div>
```

Step 3: Rendering JSON data in JavaScript File Manager

Then, map the necessary JSON data to the fileSystemData property of the File Manager using the FileData type.

Refer to the following code example.

```
import{FileManager, Toolbar, NavigationPane, DetailsView, ContextMenu, FileData}
FileManager.Inject(Toolbar, NavigationPane, DetailsView, ContextMenu);
let resultData : FileData[] = [
{
    dateCreated : new Date("2023-11-15T19:02:02.3419426+05:30"),
    dateModified : new Date("2024-01-08T18:16:38.4384894+05:30"),
    filterPath : "",
    hasChild : true,
    id : '0',
    isFile : false,
    name : "Files",
    parentId : null,
    size : 1779448,
    type : "folder",
},
{
    dateCreated : new Date("2023-11-15T19:02:02.3419426+05:30"),
    dateModified : new Date("2024-01-08T16:55:20.9464164+05:30"),
    filterPath : "\\",
    hasChild : false,
    id : '1',
    isFile : false,
    name : "Documents",
    parentId : '0',
    size : 680786,
    type : "folder",
},
{dateCreated : new Date("2023-11-15T19:02:02.3419426+05:30"), dateModified : new
```

```
    dateCreated : new Date("2023-11-15T19:02:02.3419426+05:30"),
    dateModified : new Date("2024-01-08T16:55:20.9464164+05:30"),
    filterPath : "\\Pictures\\",
    hasChild : false,
    id : '15',
    isFile : false,
    name : "Employees",
    parentId : '4',
    size : 237568,
    type : "folder",
  },
  {dateCreated : new Date("2023-11-15T19:02:02.3419426+05:30"), dateModified : n
];
let fileObject : FileManager = new FileManager({
  fileSystemData : [].slice.call(resultData) as{[key:string] : Object}[],
});
fileObject.appendTo('#filemanager');
```

Step 4: Configuring permissions

Lastly, use the Permission property to enable or restrict permission for a specific file or folder. Refer to the following code example.

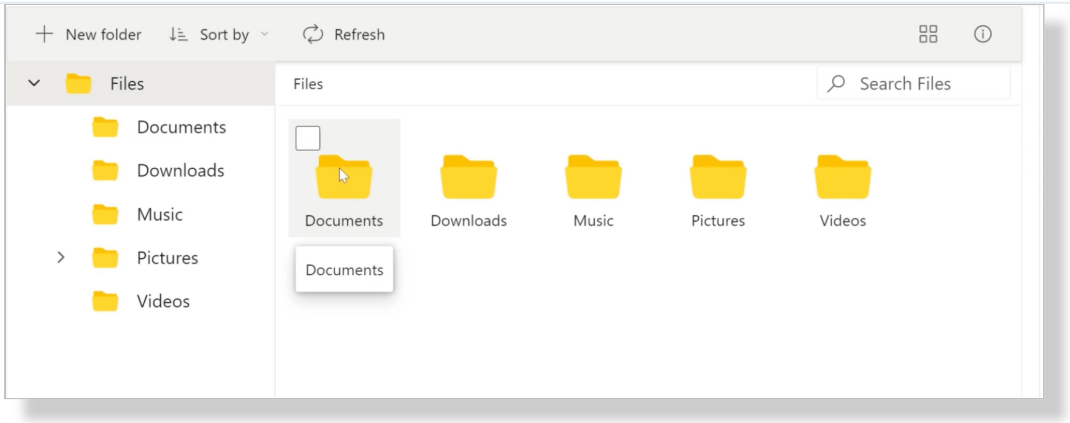
Copy

```
import {FileManager, Toolbar, NavigationPane, DetailsView, ContextMenu, Permissi
FileManager.Inject(Toolbar, NavigationPane, DetailsView, ContextMenu);

let permission: Permission = {
  "copy": false,
```

```
    "download": false,
    "write": false,
    "writeContents": false,
    "read": true,
    "upload": false,
    "message": ""
};
let resultData: FileData[] = [{
    dateCreated: new Date("2023-11-15T19:02:02.3419426+05:30"),
    dateModified: new Date("2024-01-08T16:55:20.9464164+05:30"),
    filterPath: "\\",
    hasChild: false,
    id: '1',
    isFile: false,
    name: "Documents",
    parentId: '0',
    size: 680786,
    type: "folder",
    permission: permission
}]
```

Upon completing these steps, the File Manager will be rendered with JSON data and ready to perform file operations.



Rendering flat JSON data in JavaScript File Manager

Note: For more details, refer to rendering flat JSON data in the JavaScript File Manager demo and documentation.

Handling file operations with Google Drive

By default, the JavaScript File Manager component handles file operations like creating a new folder, deleting, cutting, copying, pasting, and renaming. Certain file operations like upload, download, and get image must be dealt with the services through corresponding file action events.

Upload

To enable the upload operation in the JavaScript File Manager component with flat data, you must handle your service through the uploadListCreate event. This event provides access to the details of the file selected in the browser, including metadata such as the file name, size, and content type.

In the following code example, the File Manager retrieves the Google Drive file details as flat data for the initial rendering and uploads a file to Google Drive with the help of the **uploadListCreate** event.

Copy

```
uploadListCreate: async
function uploadFile(args) {
  var fileObj = document.getElementById("file").ej2_instances[0];
  var pathArray = fileObj.pathNames;
  var folderName = pathArray[pathArray.length - 1];
  var parentFolderId = fileObj.fileSystemData.filter(function(obj) {
    return obj.name == folderName;
  })[0].originalID;
  var folders = args.fileInfo.name.split('/');
  var fileName = folders.length & gt;
  1 ? folders[folders.length - 1] : args.fileInfo.name;
  const file = args.fileInfo.rawFile;
  // Create a new Drive API request to upload a file.
  var body = {
    'name': fileName,
    'mimeType': args.fileInfo.type,
    'parents': [parentFolderId]
```

```
    };
    var request = gapi.client.drive.files.create({
      'resource': body
    });
    request.execute(function(resp) {
      if (resp.error) {
        // Handle the error.
        console.error('Error:', resp.error.message);
        args.element.getElementsByClassName("e-file-status")[0].innerText = "Uploa
        args.element.getElementsByClassName("e-file-status")[0].classList.add("e-u
      } else {
        // Success: load the uploaded data within the File Manager component.
        args.element.getElementsByClassName("e-file-status")[0].innerText = "Uploa
        args.element.getElementsByClassName("e-file-status")[0].classList.add("e-u
        fetchData();
      }
    });
  },
```

Download

To enable the download operation in the File Manager component with flat data, you must handle your service through the beforeDownload event. This event provides access to the details of the file selected in the File Manager.

In the following code example, the File Manager retrieves the Google Drive file details as flat data for the initial rendering. Setting the **cancel** property to **true** in the **beforeDownload** event prevents the File Manager component's default delete

action. Then, you can make a Google API request with an event argument to download the raw file from Google Drive.

Copy

```
beforeDownload: function beforeDownload(args) {
  // Cancel the default download action.
  args.cancel = true;
  var fileData = args.data.data;
  const zip = new JSZip();
  //To download multiple files as a zip folder.
  if (fileData.length & gt; 1 || !fileData[0].isFile) {
    downloadFiles(fileData);
  }
  //To download a single file.
  else {
    // Fetch the file content using the Google Drive API.
    fetch(`https: //www.googleapis.com/drive/v3/files/${fileData[0].id}?alt=medi
      method: 'GET', headers: {
       'Authorization': 'Bearer ' + gapi.auth.getToken().access_token,
      },
    })
    .then(function(response) {
      if (!response.ok) {
        throw new Error('Network response was not ok: ' + response.statusText);
      }
      return response.blob();
    })
    .then(function(blob) {
       // Display image preview.
       var img = document.createElement('img');
```

```
        img.src = URL.createObjectURL(blob);
        img.alt = fileData[0].name; // Set alternative text.
        document.body.appendChild(img);

        // Create a download link.
        var downloadLink = document.createElement('a');
        downloadLink.href = URL.createObjectURL(blob);
        downloadLink.download = fileData[0].name; // Set the desired file name.
        document.body.appendChild(downloadLink);
        downloadLink.click();

        // Remove the link and image from the document.
        document.body.removeChild(downloadLink);
        document.body.removeChild(img);
    }).
    catch(function(error) {
      console.error('Error downloading file:', error);
    });
  }
},

function downloadFiles(files) {
  const zip = new JSZip();
  const totalCount = files.some(file = > file.type === "") ? getTotalFileCount(f
  const name = files.some(file = > file.type == "") ? 'folders': 'files';
  // Iterate through files and add them to the zip.
  files.forEach(file = > {
    if (file.type === '') {
      // If it's a folder, recursively fetch its contents.
      fetchFolderContents(file.id).then(response = > {
        downloadFiles(response.result.files);
      });
    } else {
```

```
      // If it's a file, download and add it to the zip.
      fetch(`https: //www.googleapis.com/drive/v3/files/${file.id}?alt=media`, {
      method: 'GET', headers: {
        'Authorization': 'Bearer ' + gapi.auth.getToken().access_token,
      },
    })
    .then(response = > {
      if (!response.ok) {
        throw new Error('Network response was not ok: ' + response.statusText);
      }
      return response.blob();
    })
    .then(blob = > {
      // Add file content to the zip.
      zip.file(file.name, blob);

      // Check if all files are added, then create the zip.
      if (Object.keys(zip.files).length === totalCount) {
        zip.generateAsync({ type: 'blob' }).then(zipBlob = > {
          // Trigger download
          const a = document.createElement('a');
          a.href = URL.createObjectURL(zipBlob);
          a.download = name + '.zip';
          document.body.appendChild(a);
          a.click();
          document.body.removeChild(a);
       });
      }
    }).
    catch(error = > {
      console.error('Error downloading file:', error);
    });
  }
```

```
  });
}
```

Get image

To enable the image preview in the File Manager component with flat data, you can use the File Manager fileSystemData property response with the **imageUrl** field.

In the following code example, the File Manager retrieves the Google Drive file details as a flat data JSON object and updates the **imageUrl** field with the Google Drive files thumbnailLink during initial rendering.

Copy

```
async function fetchData() {
  // Load the Drive API client library.
  await gapi.client.load('drive', 'v3');
  let nextPageToken = null;
  let allFiles = [];
  do {
    const response = await gapi.client.drive.files.list({
      pageSize: 1000,
      fields: 'nextPageToken, files(id, name, mimeType, size, parents, thumbnail
      pageToken: nextPageToken,
      q: 'trashed=false'
    });
```

```
    allFiles = allFiles.concat(response.result.files);
    nextPageToken = response.result.nextPageToken;
  } while ( nextPageToken );
  const files = allFiles;
  // Create a flat array representing parent-child relationships.
  window.fileSystemData = await createFlatData(files);
}

async function createFlatData(files) {
 ...
 await Promise.all(files.map(async file = > {
   ...
   var imageUrl = file.thumbnailLink;
   //Frame File Manager response data by retrieving the folder details from the
   if (file.name == 'Files') {
      rootId = file.id;
      fileDetails = {
        id: '0',
        name: file.name,
        parentId: null,
        isFile: file.mimeType == 'application/vnd.google-apps.folder' ? false: t
        hasChild: hasSubitems,
        size: file.size == undefined ? '0': file.size,
        filterPath: '',
        originalID: file.id
      };
   } else {
      fileDetails = {
        id: file.id,
        name: file.name,
        isFile: file.mimeType == 'application/vnd.google-apps.folder' ? false: t
        hasChild: hasSubitems,
        size: file.size == undefined ? '0': file.size,
```

```
            filterPath: file.filterPath,
            imageUrl: imageUrl,
            originalID: file.id
        };
    }
```

Note: For more details, refer to Events in JavaScript File Manager.

GitHub reference

Also, check out the complete example for rendering flat JSON data in JavaScript File Manager by fetching data from the Google Drive GitHub demo.

Conclusion

Thanks for reading! In this blog, we've seen how to render flat JSON data in the Syncfusion JavaScript File Manager. This feature is a part of the 2024 Volume 2 release and is also introduced in our Angular, React, Vue, ASP.NET Core, and ASP.NET MVC File Manager components.

We invite you to peruse our Release Notes and What's New pages to delve deeper into the plethora of other features introduced in this release.

For our existing customers, the latest version of Essential Studio® is readily accessible on the License and Downloads page. If you're new to Syncfusion, we offer a 30-day free trial to experience our components' entire range.

Should you have any questions, please contact us via our support forums, support portal, or feedback portal. We're always here to help you! Happy exploring!

Related blogs

- Effortlessly Synchronize JavaScript Controls Using DataManager
- Optimizing Productivity: Integrate Salesforce with JavaScript Scheduler
- PNPM vs. NPM vs. Yarn: What Should I Choose in 2024?
- Empower Your Data Insights: Integrating JavaScript Gantt Chart into Power BI

MEET THE AUTHOR

Keerthana Rajendran

Keerthana is a Product Manager at Syncfusion, working since 2016 in web control development in JavaScript, Angular, React, Vue, and Blazor platforms. She is passionate about web technology and develops Syncfusion’s web components, focusing on delivering products with perfection.