

Tableau Custom Extension for Export to Excel

INFOCEPTS

Version 1.1

Document History

Version	Date	Modified By	Summary of Changes
1.0	30-05-2022	Vinita Multani	Created

Table of Contents

1. Problem Statement	4
1.1. Features Required	4
2. Research and Development Performed.....	4
3. Export Excel Extension	5
3.1. Components of Export Excel Extension.....	5
3.2. Detailed Description of Steps Performed.....	6
3.2.1. Create a sample extension in typescript.....	6
3.2.2. Converting raw data to the formatted data object.....	7
3.2.3. Creating a workbook using ExcelJS library.....	8
3.2.4. Download excel file using File-Saver library.....	9
4. Setup and Installation - Running the Excel Export Extension.....	9
5. Important Links for Reference	11

1. Problem Statement

Tableau Development Team is working on, migrating the MSTR reports to Tableau; MSTR provides the feature for downloading an excel file of all the reports and dashboards with all the formatting and data as it appears in the reports or dashboards. If a dashboard has multiple reports then it downloads all the reports of that dashboard in the same excel file with multiple tabs (sheets) for each report. On the other hand excel export feature provided by Tableau has a few limitations, like

1. If dashboard has multiple reports then it provides to download any one report to excel file
2. Report formatting is not exactly replicated in the excel sheet.

So the problem here was to explore and devise a mechanism by which tableau can have the export to excel feature with maximum functionalities that MSTR provides.

1.1. Features Required

- Each report of dashboard in each tab (sheet) of a excel file.
- Replicate color, style, and other formatting in excel sheet as in the actual report.
- Download the excel file by single click on tableau server.

2. Research and Development Performed

So in order to solve the above problem various ways and approaches were explored and tried out, listed below are the approaches and their details in brief.

- **Tableau's Default Crosstab to Excel:** This is a default behaviour provided by tableau to download excel and csv files for the data of tableau report this provides the formatting of report up to some extent but this does not allows to download multiple reports.

https://help.tableau.com/current/pro/desktop/en-us/save_export_data.htm#export-crosstab-of-data-in-the-view-to-excel

- **Tableau JavaScript API:** This is used to integrate Tableau visualizations into your own web applications and use data from tableau visualizations to manipulate or analyse them in your web application. This provides a method called `exportCrossTabToExcel()` which gives us the output as given by **Tableau's Default Crosstab to Excel**, which has formatting to some extent but it also could not help for multiple report downloads.

https://help.tableau.com/current/api/js_api/en-us/JavaScriptAPI/js_api.htm

- **Tableau Extensions API:** This is an API developed by tableau community to interact with tableau dashboards and get data from it. It has various methods to fetch data from tableau dashboard.
<https://tableau.github.io/extensions-api/#>
- **Tableau Extension Gallery – Export All extension:** This is the extension provided by The Information Lab and its available in tableau's extension gallery, Extension gallery is a gallery where we can get plugins that can be used with our tableau visualizations, this export all extension downloads multiple tabs (sheets) for multiple reports in a single excel file, but the issue with this is, it downloads the excel file with sheets having raw data without any formatting or styles applied to it.
<https://extensiongallery.tableau.com/extensions/25?version=2020.3&per-page=50>

Basically all the above approaches could not provide the exact result that we needed, so we did a blend of one of the above approach with our custom logic using typescript, and a library called ExcelJS to create a custom extension for exporting the excel.

3. Export Excel Extension

Tableau's Extension API provides a method to create extensions using JavaScript or TypeScript that can be used to provide extended features to tableau, so we used this way to create our own extension for exporting the tableau's dashboard data to a excel file.

3.1. Components of Export Excel Extension

Depending on the requirements, we planned to develop the extension using the following components.

- **Tableau Extension API:** Tableau Extension API has various namespace, interface, and their methods to use, we used its dashboard interface to get the details of sheets (reports) included in the dashboard, and then for each sheet we used worksheet interface and its `getSummaryDataAsync()` method to fetch data of that particular report. With this we were able to get raw data of the reports; this data includes column heads and rows of data for each field.
- **TypeScript:** We got the data from extension API but it was raw data that needed modification like sorting of columns heads – as we needed measure columns to be at last pivoting - as the raw data had measure names and measure values as rows, which should be a column instead So we used typescript to write logics for converting the raw data into the format similar to the format that we have in the actual reports.

- **ExcelJs:** Now we have the modified data with us in typescript objects but we needed a mechanism for formatting this data with appropriate styles and colors and also generate an excel workbook from that data, so for this purpose we used a library (NPM Package) ExcelJS. It provides a lot of features to generate excel from your data and format, manipulate it.
- **FileSaver:** We used a library (NPM Package) **file-saver** for downloading the excel file.

3.2. Detailed Description of Steps Performed

3.2.1. Create a sample extension in typescript

The tableau extensions API provides a folder package which can be downloaded from their site and used for creating and running our extensions, that package has files for the developing, configuring, and running the extension, and also has a few sample extensions in both JavaScript and TypeScript, so with the help of a typescript sample extension we created our own typescript extension.

The extension has three main files

- **Html File** – Here you can add the html for your extension in our case it has the html for a button (export to excel), it also has script tags for including the script or other JS libraries if required, so here we provide the path to our JS file in a script tag. As we are writing our extension in TS the path here will be of compiled JS file which we get as a result of build command in a folder named **dist**.
- **TS File** – Here we have all our logic for getting data from tableau extension API, converting raw data to format we need, creating excel workbook from that data, formatting excel workbook as needed and downloading the workbook in excel format.
- **TREX file** – This is a manifest file, which is a XML file that describes the extension and provides information to register the extension with Tableau.

Refer this video to understand how to create an extension and use it with tableau - [Video Link](#)

Also we used following properties and methods of tableau extension API to get data of the dashboard which we later use in our JS file to create workbook using ExcelJS library as given in point 3.2.3.

- **tableau.extensions.initializeAsync():** This method is used to initialize the extension
 - **tableau.extensions.dashboardContent.dashboard.worksheets:** This is used to get the worksheets array which has all sheets used in the tableau dashboard.
 - **tableau.extensions.dashboardContent.dashboard.name:** This is used to give the name to the excel file as we require its name to be the dashboard name.
 - **worksheet.getSummaryDataAsync():** This is used to get the raw data for each sheet of worksheets array and use it for further processing.
-

3.2.2. Converting raw data to the formatted data object

We took the raw data as input and created an array of array for all the rows of data and then used a logic to pivot the data as shown in below example

Raw Data (Summary Data)

```

1  {
2    "data": [
3      [
4        {"value": "Central","nativeValue": "Central","formattedValue": "Central"},
5        {"value": "Red","nativeValue": "Red","formattedValue": "Red"},
6        {"value": "[sum:Unit Price:qk]","nativeValue": "[sum:Unit Price:qk]","formattedValue": "Unit Price"},
7        {"value": 234833.52999999753,"nativeValue": 234833.52999999753,"formattedValue": "234,833.53"}
8      ],
9      [
10       {"value": "Central","nativeValue": "Central","formattedValue": "Central"},
11       {"value": "Red","nativeValue": "Red","formattedValue": "Red"},
12       {"value": "[sum:Product Base Margin:qk]","nativeValue": "[sum:Product Base Margin:qk]","formattedValue": "Product Base Margin"},
13       {"value": 1473.679999999985,"nativeValue": 1473.679999999985,"formattedValue": "1,473.68"}
14     ]
15   ],
16   "columns": [
17     {"fieldName": "Region","dataType": "string","isReferenced": true,"index": 0},
18     {"fieldName": "Threshold", "dataType": "string","isReferenced": true,"index": 1},
19     {"fieldName": "Measure Names","dataType": "string","isReferenced": true,"index": 2},
20     {"fieldName": "Measure Values","dataType": "float","isReferenced": true,"index": 3}
21   ]
22 }
23

```

So from above JSON object we first create an array of '**ColumnHeaders**' using field names of each column.

```
columnNames = ['Region', 'Threshold', 'Measure Names', 'Measure Values']
```

And '**FormattedData**' from each row of data using its formattedValue.

```

formattedData = [
  ['Central','Red','Unit Price', '234,833.53'],
  ['Central','Red','Product Base Margin', '1,473.68']
];

```

So after these transformations we get the output as below

Region	Threshold	Measure Names	Measure Values
Central	Red	Unit Price	234,833.53
Central	Red	Product Base Marg	1,473.68

Thus we needed the pivoting logic, to pivot the Measure Names to Columns and show its corresponding values for each row of data.

In Pivot logic we performed the following steps

1. Loop through all data rows to find the distinct measure names, add those distinct measure names to columnNames array and then remove 'Measure Names' and 'Measure Values' from that array.

```
columnNames = ['Region', 'Threshold', 'Unit Price', 'Product Base Margin']
```

2. Loop through all data rows and created the pivot data by
 - a. First inserting the attributes of each row
 - b. Then check the measure name, get its index in the columnNames array and add the corresponding measure value at that index.

```
finalPivotArray = [
    ['Central', 'Red', '234,833.53', '1,473.68']
];
```

So we finally get the output as below

Region	Threshold	Unit Price	Product Base Margin
Central	Red	234,833.53	1,473.68

3.2.3. Creating a workbook using ExcelJS library

So from above steps we have the final pivoted data ready, now we need to create an excel workbook using that data, thus for this we used following methods of ExcelJS library.

To create a workbook:

```
const workbook = new ExcelJS.Workbook();
```

Then we add sheets to this workbook for each worksheet of worksheets array that we got in step 3.2.1 using 'tableau.extensions.dashboardContent.dashboard.worksheets' property of tableau extension API by using below method

```
let sheet = workbook.addWorksheet(worksheet.name);
```

Then we add column headers to this sheet

```
sheet.insertRow(rowNum, columnNames);
```

Then for each array of pivoted array (that we got in above step) we add rows to this sheet

```
sheet.insertRow(rowNum, rowData);
```


3.2.4. Download excel file using File-Saver library

Finally we download the workbook created in above step as an excel file using file-saver library

```
workbook.xlsx.writeBuffer().then((data: BlobPart) => {
    const blob = new Blob([data],
        { type: 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet' });
    fs.saveAs(blob, dashboardName + '.xlsx');
});
```

So finally by performing all above steps we achieve the output we needed, we are able to download an excel file named with dashboard name having tabs (sheets) for each reports used in tableau dashboard with all sort of formatting and styling.

4. Setup and Installation - Running the Excel Export Extension

To run the above extension in tableau desktop on local machine, we will need to perform few steps, the pre-requisite is to have Node.JS installed in our system

1. **Install Node.js:** Install Node.js on the your machine where you want to run the extension using the link below

Node JS: <https://nodejs.org/en/download/>

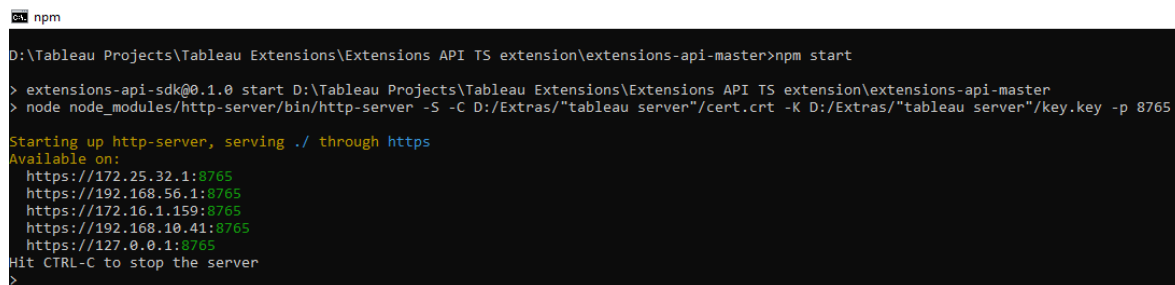
2. **Run following commands: (one by one)**

npm install

npm run build

npm start

If **npm start** is success following output will show in command prompt



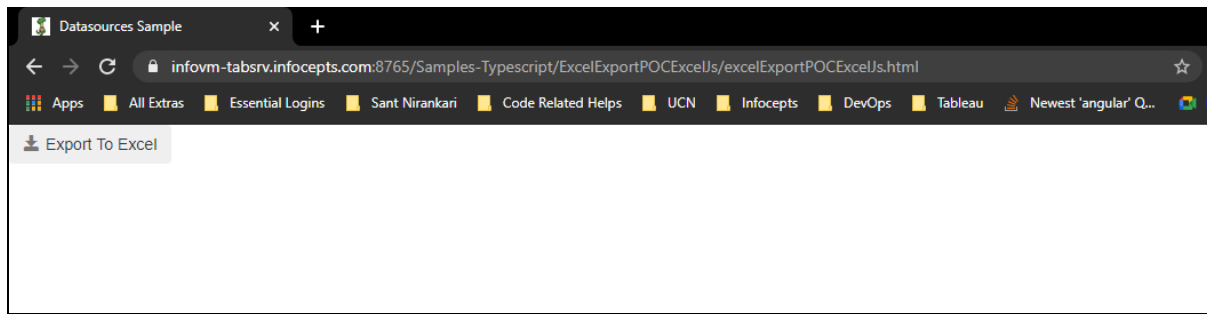
```
npm
D:\Tableau Projects\Tableau Extensions\Extensions API TS extension\extensions-api-master>npm start
> extensions-api-sdk@0.1.0 start D:\Tableau Projects\Tableau Extensions\Extensions API TS extension\extensions-api-master
> node node_modules/http-server/bin/http-server -S -C D:/Extras/"tableau server"/cert.crt -K D:/Extras/"tableau server"/key.key -p 8765

Starting up http-server, serving ./ through https
Available on:
  https://172.25.32.1:8765
  https://192.168.56.1:8765
  https://172.16.1.159:8765
  https://192.168.10.41:8765
  https://127.0.0.1:8765
Hit CTRL-C to stop the server
>
```

3. **Check if the above installation is success by hitting the below url**

<https://localhost:8765/Samples-TypeScript/ExcelExportPOCExcelJs/excelExportPOCExcelJs.html>

If you get the output as shown in image below, the installation is successful.



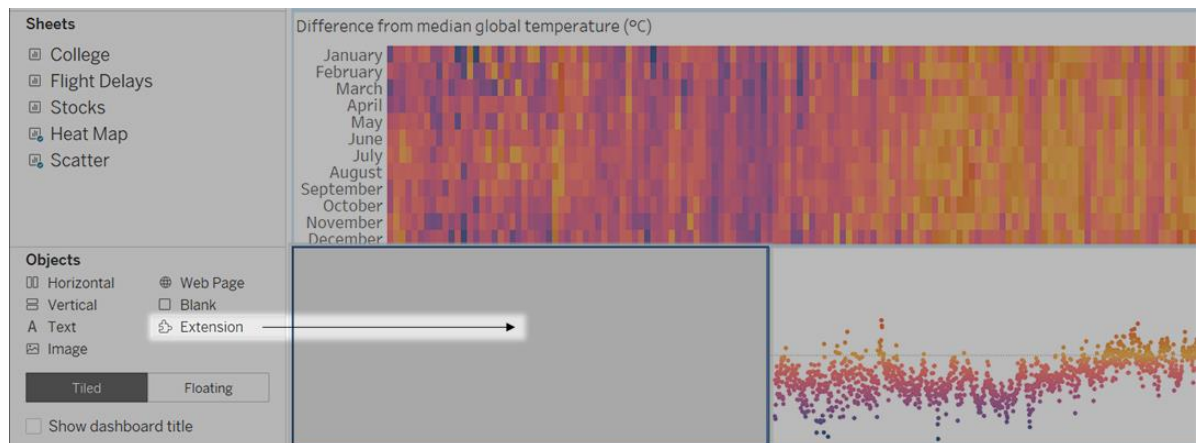
4. Configure the extension URL in .trex file in the extension

- a. Open the trex file of our extension in any editor and in the url tag add the below url

<https://localhost:8765/Samples-Typescript/ExcelExportPOCExcelJs/excelExportPOCExcelJs.html>

5. Add the extension to tableau desktop

- a. In a Tableau workbook, open a dashboard sheet.
- b. From the **Objects** section, drag **Extension** to the dashboard.



- c. In the “Add an Extension” dialog box, do either of the following:
Search for and select an extension.
- d. Click **Access Local Extensions** and navigate to a .trex file available on the following path
`<project-path>\extensions-api-master\Samples-Typescript\ExcelExportPOCExcelJs\excelExportPOCExcelJs.trex`
- e. If prompted, choose to allow, or deny the dashboard extension access to data in the workbook
- f. Safelist the URL that we generated and added to the trex file
`https://<server-domain-name>:8765/Samples-Typescript/ExcelExportPOCExcelJs/excelExportPOCExcelJs.html`
- g. Publish the workbook to the server.

6. **Using the extension:** On the published dashboard we get the button for 'Export to Excel', click the button to download the excel file.

You can check the below links for more details on adding extension and safe listing the extension URL:

[Adding Extension To Tableau Desktop](#)

[Safe listing the URL for extension](#)

5. Important Links for Reference

- ExcelJs - <https://www.npmjs.com/package/exceljs>
- File-Saver - <https://www.npmjs.com/package/file-saver>
- TypeScript – <https://www.typescriptlang.org/>
- Create and Use Extension - <https://www.youtube.com/watch?v=Mgl5lWy5lws>
- Debug Extension - https://www.youtube.com/watch?v=1qswvPhd8Yk&feature=emb_logo
- Tableau Extensions API - <https://tableau.github.io/extensions-api/#>
- Extensions API Reference Docs - https://tableau.github.io/extensions-api/docs/trex_getstarted.html