Pivot Table & Charts Component Documentation

by Flexmonster

1. API reference (/api/)

2. Getting Started

2.1. Getting started

Check out the quick-start guides to integrate Flexmonster with your framework in 5 minutes!

Get started with JavaScript

JavaScript

Get started with Angular

Angular

Get started with React

React

Get started with Vue

Vue

Explore our documentation

Data sources

See how to connect Flexmonster to your data source

Report

Configure the component as you wish and save its state for the later use

Charts

Analyze your data with our pivot charts or any other charting library

Samples

Try more than 400 examples showing how to use Flexmonster

Customization

Check out how to customize the component's appearance

Updating

Update Flexmonster to the latest version with our step-by-step guides

2.2. Get Flexmonster

We suggest different ways of including Flexmonster in your project. This guide describes all of them:

- via Flexmonster CLI (#via-cli)
- from CDN (#from-cdn)
- from the download package (#from-download-package)

Download Flexmonster via Flexmonster CLI

The most convenient way to get Flexmonster Pivot is Flexmonster CLI (/doc/cli-overview) — a command-line interface tool for Flexmonster. Install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

After that, a new flexmonster command will be available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

Now it's time to get Flexmonster Pivot.

Get Flexmonster for your project by running the appropriate command from the folder with package.json:

Pure JavaScript

```
flexmonster add flexmonster
```

This command will download Flexmonster to node_modules/ and add it as a dependency to the package.json file.

Now it’s time to start using Flexmonster in your project – see our quick start guide (/doc/how-to-create-js-pivottable/).

Angular

```
flexmonster add ng-flexmonster
```

This command will download the Flexmonster Angular module to node_modules/ and add it as a dependency to the package.json file.

Now it’s time to start using Flexmonster in your Angular project – see our guide (https://www.flexmonster.com/doc/integration-with-angular/).

React

```
flexmonster add react-flexmonster
```

This command will download the Flexmonster React module to node_modules/ and add it as a dependency to the package.json file.

Now it’s time to start using Flexmonster in your React project – see our guide (https://www.flexmonster.com/doc/integration-with-react/).

Vue

```
flexmonster add vue-flexmonster
```

This command will download the Flexmonster Vue module to node_modules/ and add it as a dependency to the package.json file.

Now it’s time to start using Flexmonster in your Vue project – see our guide (https://www.flexmonster.com/doc/integration-with-vue/).

Include Flexmonster from CDN

Get Flexmonster easily by including the files directly from our CDN.

Use the latest version of the component:

```
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>
```

Use the exact version of the component:

```
<script src="https://cdn.flexmonster.com/2.8.16/flexmonster.js"></script>
```

To learn how to embed Flexmonster in your web page, refer to our quick start guide (https://www.flexmonster.com/doc/how-to-create-js-pivottable/).

Download the Flexmonster package

The most convenient way to get Flexmonster Pivot is Flexmonster CLI (/doc/cli-overview) — a command-line interface tool for Flexmonster. Install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

Now download the Flexmonster Pivot package with the following CLI command:

```
flexmonster create javascript
```

This command will download the .zip archive with Flexmonster Pivot and automatically unpack the files in the current folder.

As a result, the flexmonster-javascript-project/ folder will appear in your working directory. Open it and copy the flexmonster/ folder into the root of your web project.

Note: to use a trial version of Flexmonster, you will need to set a trial key. It can be found in the flexmonster-javascript-project/TrialKey.txt file. Copy the contents of the file and set the trial key via the licenseKey initialization parameter:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

To embed Flexmonster in your project, see the quick start guide (https://www.flexmonster.com/doc/how-to-create-js-pivottable/).

2.3. Quick start

Get started with Flexmonster Pivot Table & Charts in 5 minutes!

To use Flexmonster with frameworks, see our guides on integrations with:

- Angular (https://www.flexmonster.com/doc/integration-with-angular/)
- React (https://www.flexmonster.com/doc/integration-with-react/)
- Vue (https://www.flexmonster.com/doc/integration-with-vue/)
- many other frameworks (https://www.flexmonster.com/doc/integration/)

Initialize Flexmonster

Embed Flexmonster Pivot into your web page in 4 simple steps. If you have any issues, visit our troubleshooting page (https://www.flexmonster.com/doc/typical-errors/).

Step 1. Add a container for Flexmonster

Create a <div> container for the component:

```
<div id="pivotContainer">The component will appear here</div>
```

Step 2. Include Flexmonster in the HTML page

If needed, get Flexmonster (https://www.flexmonster.com/doc/get-flexmonster/) and then include it in your web page with one of the following scripts:

Include from the npm package

```
<div id="pivotContainer">The component will appear here</div>
<script src="node_modules/flexmonster/flexmonster.js"></script>
```

Include from CDN

```
<div id="pivotContainer">The component will appear here</div>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>
```

Include from the download package

```
<div id="pivotContainer">The component will appear here</div>
<script src="flexmonster/flexmonster.js"></script>
```

Step 3. Embed the component

Add a simple script to embed the component into the web page:

For the npm package

```
<div id="pivotContainer">The component will appear here</div>
<script src="node_modules/flexmonster/flexmonster.js"></script>

<script>
    var pivot = new Flexmonster({
        container: "pivotContainer",
        componentFolder: "node_modules/flexmonster/",
        toolbar: true
    });
</script>
```

Notice the componentFolder parameter – it should point to the flexmonster/ folder. Since Flexmonster is included via npm, componentFolder should be defined as node_modules/flexmonster/.

For CDN

```
<div id="pivotContainer">The component will appear here</div>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>

<script>
    var pivot = new Flexmonster({
        container: "pivotContainer",
        componentFolder: "https://cdn.flexmonster.com/",
        toolbar: true
    });
</script>
```

Notice the componentFolder parameter – it should point to the flexmonster/ folder. Since Flexmonster is included from CDN, componentFolder should be defined as https://cdn.flexmonster.com/.

For the download package

```
<div id="pivotContainer">The component will appear here</div>
<script src="flexmonster/flexmonster.js"></script>

<script>
    var pivot = new Flexmonster({
        container: "pivotContainer",
        componentFolder: "flexmonster/",
        toolbar: true
    });
</script>
```

Flexmonster has more initialization parameters — check them out (https://www.flexmonster.com/api/new-flexmonster/).

Step 4. See the result

Now launch the page from a browser — the component without data is embedded into your project. You can see the live example on JSFiddle (https://jsfiddle.net/flexmonster/L54jrsp5/).

Troubleshooting

If you have any issues, visit our troubleshooting page (https://www.flexmonster.com/doc/typical-errors/).

Next steps

Visit our detailed guides and learn different aspects of using the component:

- How to connect to your data source (https://www.flexmonster.com/doc/supported-data-sources/)
- How to configure the Flexmonster report (https://www.flexmonster.com/doc/configuring-report/)
- How to export the data from the component (https://www.flexmonster.com/doc/export-and-print/)
- How to customize the appearance (https://www.flexmonster.com/doc/customizing-appearance/)
- How to integrate with charts (https://www.flexmonster.com/doc/available-tutorials-charts/)

2.4. System requirements

- A web browser. It is recommended that you use the most up-to-date version available for the best experience. The minimum browser requirements are listed below:
    - Chrome 12+
    - Firefox 15+
    - Internet Explorer 11 (see usage note (#ie11))
    - Microsoft Edge
    - Opera 15+
    - Safari 6.1+
    - iOS Safari 5.1.1+
- JavaScript must be enabled.
- The minimal recommended size for the pivot table component is 400×300px.

Internet Explorer 11 support

Note that starting from version 2.8.9, flexmonser.js was migrated to ES6 JavaScript standard, which is not supported by Internet Explorer 11. For Internet Explorer 11, we made a special ES5 version flexmonster.es5.js. It is available:

- On the CLI and npm (along with flexmonster.es5.full.js)
- On CDN (https://cdn.flexmonster.com/flexmonster.es5.js)
- In download packages (inside the flexmonster/ folder)

2.5. Troubleshooting

Welcome to our troubleshooting page. Here you can find an explanation of errors that you might experience while working with Flexmonster Pivot, as well as simple instructions on how to fix them. If your error is not listed here, post a question to our Help Forum (/forum/).

This page has three sections:

1. Installation troubleshooting (#installation-troubleshooting)
2. Issues with license keys (#license-keys)
3. Issues with data source (#data-source)

Installation troubleshooting

If you are facing any problems with the embedding of the component, in your browser go to the page where Flexmonster should be displayed and open the browser's console to check if there are any errors in the console. This section provides solutions to the errors that you may see in the console.

Console error: 'GET (any URL)/flexmonster.js 404 (Not Found)' (#!flexmonster-not-included)

This error means that flexmonster.js was not loaded successfully. Ensure that the correct path is specified to the flexmonster.js file:

For the npm package

```
<script src="node_modules/flexmonster/flexmonster.js"></script>
```

For CDN

```
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>
```

For the download package

```
<script src="flexmonster/flexmonster.js"></script>
```

Console error: 'Uncaught ReferenceError: Flexmonster is not defined' (#!flexmonster-not-defined)

Getting this error means that the new Flexmonster() API call was used to embed the component but flexmonster.js was not loaded successfully. Make sure that you have included flexmonster.js in your HTML page:

For the npm package

```
<script src="node_modules/flexmonster/flexmonster.js"></script>
```

For CDN

```
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>
```

For the download package

```
<script src="flexmonster/flexmonster.js"></script>
```

Console error: 'SCRIPT5009: Flexmonster is undefined' (#!flexmonster-undefined-ie)

Getting this error in Internet Explorer means that the ES6 version of the component is used. Internet Explorer does not support the ES6 standard (/doc/system-requirements/#!ie11), so use the ES5 version of the component (flexmonster.es5.js) for this browser. It is available:

- On the CLI and npm (along with flexmonster.es5.full.js)
- On CDN (https://cdn.flexmonster.com/flexmonster.es5.js)
- In download packages (inside the flexmonster/ folder)

Console error: 'ERROR TypeError: Flexmonster is not a constructor' (#!flexmonster-not-constructor)

Such an error means that Flexmonster embedding failed since flexmonster.js was not loaded successfully. Make sure that the flexmonster.js file is successfully loaded in your project:

For the npm package

```
<script src="node_modules/flexmonster/flexmonster.js"></script>
```

For CDN

```
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>
```

For the download package

```
<script src="flexmonster/flexmonster.js"></script>
```

Console error: 'ERROR TypeError: window.Flexmonster is not a function' (#!flexmonster-not-function)

This error indicates that an attempt to embed Flexmonster failed because flexmonster.js was not loaded successfully. To resolve this error, include this file in your project:

For the npm package

```
<script src="node_modules/flexmonster/flexmonster.js"></script>
```

For CDN

```
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>
```

For the download package

```
<script src="flexmonster/flexmonster.js"></script>
```

Pop-up alert and console errors: 'Flexmonster: Unable to create the component. DOM element is null.' (#!no-container-parameter)

This error is thrown if the container parameter is missing in the new Flexmonster() API call. This parameter is necessary because it sets the selector of the HTML element which will be the container for the component. For an example of how container should be specified, refer to step 3 of our Quick start guide (/doc/how-to-create-js-pivottable/#embed-component).

Pop-up alert and console errors: 'Flexmonster: Unable to create the component. DOM element with id 'pivotContainer' is not found.' (#!no-pivot-container)

Such an error indicates that the <div> container for the component was not created. Add a container to your HTML page like so:

```
<div id="pivotContainer">The component will appear here</div>
```

Pop-up alert error: 'Flexmonster: Pivot cannot be drawn.' (#!cannot-be-drawn)

This error means that the content of the flexmonster/ folder was not loaded successfully. Add the componentFolder parameter to the new Flexmonster() call. For more details about this parameter refer to the new Flexmonster() (/api/new-flexmonster/) API call.

Console error: 'GET (any URL)/flexmonster/flexmonster.css 404 (Not Found)' (#!no-component-folder)

This error means that the component is located in a folder other than flexmonster/. Specify the componentFolder parameter. For more details about this parameter refer to the new Flexmonster() (/api/new-flexmonster/) API call.

Console error: 'GET (any URL)/(your componentFolder parameter)/flexmonster.css 404 (Not Found)' (#!wrong-component-folder)

This error indicates that you specified the componentFolder parameter incorrectly. Make sure that componentFolder contains the URL of the component’s folder (with flexmonster.css and all other necessary files).

Console error: 'GET (any URL)/(your componentFolder parameter)/theme/assets/flexmonster-icons.woff 404 (Not Found); GET (any URL)/(your componentFolder parameter)/theme/assets/flexmonster-icons.ttf 404 (Not Found)' (#!assets-folder)

This error means that the theme/assets/ folder is missing from the component’s folder. Add this folder, which can be found in the flexmonster/ folder inside the download package.

Console error: 'GET (any URL)/(your componentFolder parameter)/toolbar/flexmonster.toolbar.js 404 (Not Found)' (#!toolbar-folder)

This error indicates that the toolbar/ folder is missing from the component’s folder and that toolbar: true was specified when embedding the component. Add this folder, which can be found in the flexmonster/ folder inside the download package.

Console error: 'GET (any URL)/(your componentFolder parameter)/lib/d3.min.js 404 (Not Found)' (#!lib-folder)

This error means that the lib/ folder is missing from the component’s folder. The error is also possible if the d3.min.js file was manually removed from the lib/ folder. The d3.min.js library is necessary for using any charts functionality, so the error will only be shown when switching to charts. If any other file was removed from lib/ folder, the same error will appear in the console (only when the component needs these libraries). To get rid of such errors, add the lib/ folder to the component's folder. The lib/ folder can be found in the flexmonster/ folder inside the download package.

License keys

This section explains the meaning of the error pop-ups you may experience if a problem occurs with license keys.

Current key is only applicable for example.com. You are trying to use the following key: XXXX-XXXX-XXXX-XXXX-XXXX (#!current-key)

Verify that the domain name shown in your error message (e.g. example.com) matches the domain name of your project for which you have the key. If they are different, contact our team (/contact/).

Integration with third-party charting libraries is not available in the trial version. (#!third-party)

For our existing customers, we recommend replacing their trial license key with a development license key or with a production license key. If you are evaluating our component, please contact our team (/contact/) and request a special trial key.

Your license key is outdated and will not function with the current version. Please contact our support team to find out about upgrade options. (#!outdated-key)

If you are updating from the previous major version to 2.3 or higher, new license keys are required. If your maintenance is active – please contact our team (/contact/).

You are trying to use a developer's key on a real domain (example.com). You are trying to use the following key: XXXX-XXXX-XXXX-XXXX-XXXX (#!dev-key)

This message indicates that a development license key is used which is applicable to a localhost environment only. To get the production key for the production environment or the development key for the real domain (e.g., example.com), please contact our team (/contact/).

Your license period has expired. You are trying to use the following key: XXXX-XXXX-XXXX-XXXX-XXXX (#!license-expired)

This message is shown if your license key has expired. To renew your annual subscription, please contact our team (/contact/).

Your trial period has expired. You are trying to use the following key: XXXX-XXXX-XXXX-XXXX-XXXX (#!trial-expired)

This message means your trial license key has expired. To continue the evaluation of our component, please contact our team (/contact/?r=tex) for trial extension.

Invalid license key. You are trying to use the following key: XXXX-XXXX-XXXX-XXXX-XXXX (#!invalid-license-key)

Copy the license key you received after purchase and use it in your project. Also, check that the license key version is the same as the component’s version. For example, a license key for version 2.3 will not work with previous versions.

You are trying to use a template license or trial key: "XXXX-XXXX-XXXX-XXXX-XXXX". Please replace "XXXX-XXXX-XXXX-XXXX-XXXX" with an actual key. (#!key-placeholder)

This message means that you are using a key placeholder instead of a real license or trial key. Please replace "XXXX-XXXX-XXXX-XXXX-XXXX" with your actual key. If you don’t have a license key, please contact our team (/contact/).

License key not found. (#!license-number-not-found)

This message means that you have not specified a license key. Please specify your license key. For instructions on how the license key should be specified, refer to our guide on the license keys (/doc/managing-license-keys/#!set-license-key).

License key is corrupted. (#!license-number)

This message means that you have probably copied only a part of the key. Please open the message you received after purchase and try copying the license key again.

Serial number not found. (#!serial-number-not-found)

This message means that you have not specified a license key. Please specify your license key. For instructions on how the license key should be specified, refer to our guide on the license keys (/doc/managing-license-keys/#!set-license-key).

Serial number is corrupted. (#!serial-number)

This message means that you have probably copied only a part of the key. Open the message you received after purchase and try copying the license key again.

Data source

This section explains the meaning of errors you may experience if a problem occurs with the data source.

Unable to open file 'yourfile'. It seems that this file doesn't exist or 'Access-Control-Allow-Origin' header is absent in the requested resource. (#!unable-to-open-file)

This message may refer to one of the reasons below:

- There is no file with this name on the server. Make sure the filename you are trying to use is correct.
- The browser requires more privileges to load the data. By default, cross-domain requests from JavaScript are blocked. Enable CORS to make such requests. Refer to enable-cors.org (https://enable-cors.org/) for more details on how to resolve this issue.
- Internal server error — please open the browser's console and check the errors there.

CSV and JSON support is not available in the current edition. (#!csv-json)

Each license key is bound to a data source. Check that the data source type for your license key, the type of the downloaded package, and the data source type you are using are the same. If you want to test a data source that is not included in your licensing plan, contact our team (/contact/).

MS OLAP support is not available in the current edition. (#!msas)

Each license key is bound to a data source. Check that the data source type for your license key, the type of the downloaded package, and the data source type you are using are the same. If you want to test a data source that is not included in your licensing plan, contact our team (/contact/).

Elasticsearch support is not available in the current edition. (#!es)

Each license key is bound to a data source. Check that the data source type for your license key, the type of the downloaded package, and the data source type you are using are the same. If you want to test a data source that is not included in your licensing plan, contact our team (/contact/).

Mondrian support is not available in the current edition. (#!mondrian)

Each license key is bound to a data source. Check that the data source type for your license key, the type of the downloaded package, and the data source type you are using are the same. If you want to test a data source that is not included in your licensing plan, contact our team (/contact/).

icCube support is not available in the current edition. (#!iccube)

Each license key is bound to a data source. Check that the data source type for your license key, the type of the downloaded package, and the data source type you are using are the same. If you want to test a data source that is not included in your licensing plan, contact our team (/contact/).

File is too large. (#!large-file)

This message means that your Flexmonster edition is Pivot Table for SQL/CSV/JSON Basic. This edition has a 5 MB limitation on data size and your file exceeds this limitation. To upload bigger files, upgrade to the SQL/CSV/JSON edition. For further details please contact our team (/contact/).

Error opening URL. Please check your Internet connection. (#!url-error)

This error means one of the following:

- There is no Internet connection. Reestablish this connection.
- You provided the wrong path to the proxy URL to the OLAP data source (Microsoft Analysis Services or Mondrian). Check that the URL is correct.
- There is an issue with accessing the data — make sure that current user has enough privileges.
- A server error occurred — check whether you have any errors in the browser's console.

Invalid datasource or catalog. Please check. (#!datasource)

Check the catalog and cube names that you entered for the connection to the cube. To get the exact names, use our OLAP connection tool from the Toolbar. Click the Connect tab, select To OLAP (XMLA), enter your Proxy URL, and click Connect. Select Data Source Info from the drop-down list. The list of available catalog names will be right under Data Source Info. After choosing the catalog name, the list of cube names will be shown. Another possible cause of this error message is an internal server error. Try opening the browser's console and checking there.

Stream error occurred while loading example.com (#!stream-error)

Check the following points:

- Check that either your filename (for CSV and JSON data sources) or the proxy URL path, catalog, and cube names (for SSAS and Mondrian data sources) exist and that the current user has sufficient rights to access them.
- Make sure that cross-domain requests are allowed. Additional information on this resource can be found here: enable-cors.org (https://enable-cors.org/).
- Open the console in your browser and check for internal server errors.
- Check your Internet connection.
- If you are using SSAS via XMLA protocol, enable cross-origin resource sharing for Internet Information Services (IIS). Check out our detailed step-by-step guide (/question/stream-error-occurred-while-loading-httplocalhost8080olapmsmdpump-dll/). We also suggest trying our special server-side proxy called Flexmonster Accelerator instead of XMLA (read more (/doc/getting-started-with-accelerator-ssas/)).

2.6. Managing license keys

All Flexmonster Pivot Table & Charts editions need a license key. This guide describes the license key types and how to use the license keys:

- Setting a license key (#set-license-key)
- Checking a license key (#check-license-key)
- Types of keys (#types-of-keys)

Setting a license key

When you get a license key, set it for the component via the licenseKey (https://www.flexmonster.com/api/new-flexmonster/) parameter. Look at the examples of setting a license key:

in pure JavaScript

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

in Angular

```
<fm-pivot
 [licenseKey]="'XXXX-XXXX-XXXX-XXXX-XXXX'">
</fm-pivot>
```

in React

```
<FlexmonsterReact.Pivot
 licenseKey="XXXX-XXXX-XXXX-XXXX-XXXX"
/>
```

in Vue

```
<Pivot
 ref="pivot"
 v-bind:licenseKey="'XXXX-XXXX-XXXX-XXXX-XXXX'">
</Pivot>
```

If you experience any error pop-up window, see the Troubleshooting guide (https://www.flexmonster.com/doc/typical-errors/#!license-keys).

Checking a license key

You can check your license key by clicking on the grid and pressing Ctrl + Alt + i (Option + Control + i on macOS). This will open the pop-up window displaying information about your license.

Types of keys

We offer the following types of keys:

- A **trial key** is used for the trial version. The component runs with a trial key **without any restrictions**, with all functionality available in the downloaded edition. The only limitation is that integration with 3rd party charting libraries is not available.
  This key **is temporary** and has an expiration date. Usually, the trial period lasts for 30 days.
  A trial key is usually **set automatically**, except for cases when the download package is used. Learn how to set a trial key for the download package (https://www.flexmonster.com/doc/get-flexmonster/#!from-download-package).
- A **development license key** is provided for development purposes. It is applicable to your localhost environment. Using a development key, the component can be run locally on your computer (**localhost**) or on a server using the server’s **IP address**.
  We do not license on a per-developer basis, so you only need one license for the whole development team that is working on the same project. This key is issued right after the purchase. A development license key can have an expiration date or it can be perpetual – depending on the purchased license.
- A **production license key** is provided for the production environment (domain/URL) where the component runs (e.g. yourcompany.com). You do not need to know the production domain at the moment of purchase, instead, you can ask for the production key just prior to publishing your application on the web.
  This key is **tied to your domain name**. A production license key can have an expiration date or it can be perpetual – depending on the purchased license.
- A **staging key** can be issued by request after a Flexmonster license is purchased. It can be used for testing on the real domain, but not in the production environment.

You will receive a development key right after you purchase a Flexmonster license. When you know the target domain/URL, you will receive a production key.

2.7. Migrating from WebDataRocks to Flexmonster

This guide will walk you through the process of transitioning from WebDataRocks to Flexmonster.

Let’s break the migration into small steps.

1. Installation (#installation)
2. Creating a pivot instance (#embedding)
3. Setting a slice (#slicing)
4. Data sources (#datasources)
5. Migrating from WebDataRocks in Angular (#angular)

Installation

All the possible ways of getting Flexmonster are described in the Get Flexmonster (https://www.flexmonster.com/doc/get-flexmonster/) guide.

Creating a pivot instance

Next, replace the WebDataRocks initialization API call (https://www.webdatarocks.com/doc/init-api-call/) with Flexmonster's (https://www.flexmonster.com/api/new-flexmonster/):

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    report: {
        dataSource: {
            filename: "data.csv"
        }
    }
});
```

Notice the differences between the init API calls of Flexmonster and WebDataRocks.

Flexmonster has an additional componentFolder parameter, which should point to the flexmonster/ folder. Read more about Flexmonster initialization parameters (https://www.flexmonster.com/api/new-flexmonster/).

Setting a slice

In WebDataRocks, we specified which hierarchies to put into the rows, columns, and measures by defining a Slice Object (https://www.webdatarocks.com/doc/slice-object/). The structure of the slice in Flexmonster (https://www.flexmonster.com/api/slice-object/) is no different, except that measures should be placed in square brackets:

```
"slice": {
    "rows": [
        { "uniqueName": "Customer" }
    ],
    "columns": [
        { "uniqueName": "Month" },
        { "uniqueName": "[Measures]" }
    ],
    "measures": [
        {
            "uniqueName": "Revenue",
            "aggregation": "sum"
        }
    ]
}
```

Data sources

Flexmonster allows you to connect to many more types of data sources. On top of CSV and JSON, you can analyze data from relational or NoSQL databases, MongoDB, Elasticsearch, SSAS, Mondrian, and custom data source API. To learn how to connect to these sources, refer to the following guides:

- Connecting to JSON (https://www.flexmonster.com/doc/json-data-source/)
- Connecting to CSV (https://www.flexmonster.com/doc/csv-data-source/)
- Connecting to relational databases (/doc/connect-to-relational-database/)
- Connecting to NoSQL databases (https://www.flexmonster.com/doc/connecting-to-other-databases/)
- Connecting to MongoDB databases (/doc/mongodb-connector/)
- Connecting to Elasticsearch (https://www.flexmonster.com/doc/connecting-to-elasticsearch/)
- Connecting to Microsoft Analysis Services (https://www.flexmonster.com/doc/connecting-to-microsoft-analysis-services/)
- Implementing the custom data source API (https://www.flexmonster.com/doc/introduction-to-custom-data-source-api/)

(https://www.flexmonster.com/doc/connecting-to-microsoft-analysis-services/)Migrating from WebDataRocks in Angular

If you are working on an Angular project, follow these steps:

Step 1. Install Flexmonster CLI (https://www.flexmonster.com/doc/cli-overview) — the most convenient way to work with Flexmonster Pivot:

```
npm install -g flexmonster-cli
```

Now a new flexmonster command is available in the console.

Step 2. Install the Flexmonster Angular module – run the following Flexmonster CLI command from the folder containing package.json:

```
flexmonster add ng-flexmonster
```

This command will install the ng-flexmonster package to node_modules/ and add it as an npm dependency to package.json.

Step 3. Add CSS and JS references to the angular.json file:

```
"styles": [
    "styles.css",
    "/node_modules/flexmonster/flexmonster.min.css"
 ],
"scripts": ["/node_modules/flexmonster/flexmonster.full.js"]
```

Step 4. Replace the WebDataRocksPivot module with FlexmonsterPivotModule in app.module.ts. app.module.ts can be found inside the PROJECT-NAME/src/app folder. It should look as follows:

```
import { FlexmonsterPivotModule } from 'ng-flexmonster';
@NgModule({
     ...
     imports: [FlexmonsterPivotModule],
     ... })
```

Step 3. Find the wbr-pivot directive that includes the pivot table and replace it with an fm-pivot directive. Here is an example:

```
<wbr-pivot
 [report]="'https://cdn.flexmonster.com/reports/report.json (https://cdn.flexmonster
.com/reports/report.json)'">
</wbr-pivot>
```

should be changed to:

```
<fm-pivot
 [report]="'https://cdn.flexmonster.com/reports/report.json (https://cdn.flexmonster
.com/reports/report.json)'">
</fm-pivot>
```

Step 3. Build and run your application from the console:

```
ng serve
```

The default port that the application listens to is 4200. Open http://localhost:4200/ in your browser to see the results.

To find out more about integrating Flexmonster Pivot with Angular, check out the Integration with Angular (https://www.flexmonster.com/doc/integration-with-angular/) guide.

3. Integration with frameworks

3.1. Available tutorials

From Flexmonster version 2.3 onwards, the component can be easily integrated with popular frameworks. It only takes a few lines of code to start using Flexmonster in your JS framework.

The pivot table can be natively used with JavaScript or TypeScript and integrates with client-side frameworks such as jQuery, Angular, AngularJS, React, React Native, Vue, Electron.js, Ionic, or Require. Flexmonster also works well with Webpack, R Shiny, and Python.

Guides

Follow the detailed tutorials for each technology:

- Integration with Angular (/doc/integration-with-angular/)
- Integration with React (/doc/integration-with-react/)
- Integration with Vue (/doc/integration-with-vue/)
- Integration with Python (Django) (/doc/integration-with-django/)
- Integration with Python (Jupyter Notebook) (/doc/integration-with-jupyter-notebook/)
- Integration with React Native (/doc/integration-with-react-native/)
- Integration with AngularJS (/doc/integration-with-angularjs/)
- Integration with TypeScript (/doc/integration-with-typescript/)
- Integration with R Shiny (/doc/integration-with-r-shiny/)
- Integration with jQuery (/doc/integration-with-jquery/)
- Integration with Ionic (/doc/integration-with-ionic/)
- Integration with Electron.js (/doc/integration-with-electron-js/)
- Integration with Webpack (/doc/integration-with-webpack/)
- Integration with Require (/doc/integration-with-requirejs/)

3.2. Integration with Angular

This tutorial will help you integrate Flexmonster with the Angular framework (https://angular.io/). It is based on angular.io quickstart (https://angular.io/docs/ts/latest/quickstart.html). Flexmonster is fully compatible with the latest Angular 11 as well as earlier 4+ versions.

Prerequisites

To run a simple application you will need Node.js and npm. Get it here (https://docs.npmjs.com/getting-started/installing-node) if it’s not already installed on your machine.

Open a terminal/console window and verify that you are running at least node v4.x.x and npm 3.x.x by running node -v and npm -v.

Then install the Angular CLI (https://cli.angular.io/) globally by running:

```
npm install -g @angular/cli
```

One more command-line interface tool needed is Flexmonster CLI (/doc/cli-overview/), which is the most convenient way to work with Flexmonster. Install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

Now a new flexmonster command is available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

After that, choose one of the following options:

1. Run a sample Angular 5+ project from GitHub (#github-sample)
2. Integrate Flexmonster into an Angular 5+ application (#integration)
3. See the examples of Flexmonster usage in Angular (#examples)
4. Integrate Flexmonser into an Angular 4 application (#integration-angular4)

Run a sample Angular 5+ project from GitHub

Create the sample project with the following CLI command:

```
flexmonster create angular -r
```

The flexmonster create angular command will do the following:

- Download the .zip archive with the sample Angular project from GitHub.
- Automatically unpack the files in the current folder — as a result, the flexmonster-angular-project/ folder will appear in your working directory.

The -r option, which stands for --run, completes these tasks:

- Installs all the npm dependencies described in package.json.
- Compiles the application and runs it in the browser.

The application can be shut down manually with Ctrl+C.

Integrate Flexmonster into an Angular 5+ application

To integrate Flexmonster into an Angular 5+ app, follow these steps:

Step 1. If you don’t have an Angular CLI app, you can create it by running these commands in the console:

```
ng new PROJECT-NAME
cd PROJECT-NAME
```

Step 2. Install the Flexmonster Angular module by running this CLI command from the folder containing package.json :

```
flexmonster add ng-flexmonster
```

The add command will install the ng-flexmonster package to node_modules/ and add it as an npm dependency to package.json.

Step 3. Import FlexmonsterPivotModule into src/app/app.module.ts:

```
import { FlexmonsterPivotModule } from 'ng-flexmonster';
```

```
@NgModule({
  ...
  imports: [FlexmonsterPivotModule],
  ...
})
```

Step 4. Import CSS styles (e.g. in the styles.css):

```
@import "flexmonster/flexmonster.min.css"
```

Step 5. Import flexmonster and ng-flexmonster TypeScript modules (e.g. in the app.component.ts):

```
import * as Flexmonster from 'flexmonster';
import { FlexmonsterPivot } from 'ng-flexmonster';
```

Step 6. Insert fm-pivot directive where you need the pivot table (e.g. in the app.component.html):

```
<fm-pivot
  [report]="'https://cdn.flexmonster.com/reports/report.json'">
</fm-pivot>
```

Step 7. Run your application from the console:

```
ng serve
```

To see the result open your browser on http://localhost:4200/.

If you want to integrate with charting libraries, we included all necessary type definitions for integration with Google Charts, FusionCharts, and Highcharts.

Examples

The sample Angular project with Flexmonster contains several usage examples (https://github.com/flexmonster/pivot-angular/tree/master/src/app/examples). You can try them all on the project's starting page.

This section gives a detailed description of each example:

- Adding the pivot table (#add-pivot-table)
- Calling events (#use-events)
- Using API calls (#use-methods)
- Updating data (#update-data)
- Customizing the Toolbar (#customize-toolbar)
- Customizing the grid (#customize-grid)
- Integrating with Highcharts (#with-highcharts)

- Integrating with amCharts (#with-amcharts)

Adding the pivot table

The first example (https://github.com/flexmonster/pivot-angular/tree/master/src/app/examples/pivot-table-demo) demonstrates the basic usage of Flexmonster. In pivot-table-demo.component.html, see how the toolbar and report initialization parameters are specified in Angular.

Flexmonster has more initialization parameters. Have a look at all of them (https://www.flexmonster.com/api/new-flexmonster/).

Calling events

This usage example (https://github.com/flexmonster/pivot-angular/tree/master/src/app/examples/calling-events) is focused on Flexmonster events. It provides a toggle button for subscribing to all the events and unsubscribing from them. See how calling the events is implemented in the calling-events.component.ts file.

When Flexmonster is subscribed to the events, the log output displays:

- Which event was triggered.
- When the event was triggered.
- Details on that event.

See the full list of Flexmonster events in our documentation (https://www.flexmonster.com/api/events/).

Using API calls

The Using API calls (https://github.com/flexmonster/pivot-angular/tree/master/src/app/examples/using-api-calls) section is about customizing the component with API calls. Switch the toggle buttons to:

- Show the pie chart.
- Show the grid.
- Make the component read-only.
- Make the component interactive.

All the functionality is implemented in using-api-calls.component.ts.

See the full list of Flexmonster API calls (https://www.flexmonster.com/api/methods/).

Updating data

The Updating data (https://github.com/flexmonster/pivot-angular/tree/master/src/app/examples/updating-data) section contains an example of data updating at runtime. The example uses the updateData() (https://www.flexmonster.com/api/updatedata/) API call in the function to update the data. See the source code in the updating-data.component.ts file.

Customizing the Toolbar

Go to the Customizing the Toolbar (https://github.com/flexmonster/pivot-angular/tree/master/src/app/examples/customizing-toolbar) section to see the example of Toolbar customization.

We use the beforetoolbarcreated event to invoke the customizeToolbar() function. As a result, a custom tab with a custom functionality is added.

The beforetoolbarcreated event is called in customizing-toolbar.component.html, and its handler can be found in the customizing-toolbar.component.ts file.

Learn more about customizing the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/).

Customizing the grid

The Customizing the grid (https://github.com/flexmonster/pivot-angular/tree/master/src/app/examples/customizing-grid) example demonstrates how the component can be customized.

Switch the toggle buttons to apply or remove customization. Custom grid styles are defined in customizeCellFunction() in the customizing-grid.component.ts file.

See our documentation (https://www.flexmonster.com/doc/customizing-grid/) to learn more about cell customizing.

Integrating with Highcharts

See an example of integration with Highcharts in the With Highcharts (https://github.com/flexmonster/pivot-angular/tree/master/src/app/examples/with-highcharts) section.

Here are the main elements of this integration:

1. The Highcharts module and Flexmonster Connector for Highcharts.
2. A container for Highcharts.
3. The reportcomplete event; when triggered, the function to draw the chart is invoked.
4. The drawChart() function; find it in the with-highcharts.component.ts file.

Integration with other charting libraries can be done in a similar way. Check out the integrations we provide (https://www.flexmonster.com/doc/available-tutorials-charts/).

Integrating with amCharts

In the With amCharts (https://github.com/flexmonster/pivot-angular/tree/master/src/app/examples/with-amcharts) section, you can see a dashboard with Flexmonster and amCharts.

The key elements of this integration are:

1. The amCharts module and Flexmonster Connector for amCharts (https://www.flexmonster.com/api/all-methods-amcharts/).
2. A container for amCharts.
3. The reportcomplete event; when triggered, the function to draw the chart is invoked.
4. The drawChart() function; find it in the with-amcharts.component.ts file.

Integration with other charting libraries can be done in a similar way. Check out the integrations we provide (https://www.flexmonster.com/doc/available-tutorials-charts/).

Integrate Flexmonser into an Angular 4 application

To integrate Flexmonster into an Angular 4 project, we recommend using the FlexmonsterPivot module from GitHub. This module is also available on npm, but it can only be used with Angular 5+ versions due to the metadata files’ restriction.

Follow the steps below:

Step 1. If needed, uninstall the ng-flexmonster npm module:

```
npm uninstall ng-flexmonster
```

Step 2. Download the FlexmonsterPivot source code from GitHub (https://github.com/flexmonster/ng-flexmonster/blob/master/ng-flexmonster/src/flexmonster.component.ts) and move it to the src/app/ folder.

Step 3. Install the flexmonster npm module:

```
npm install flexmonster
```

Step 4. In the app.component.ts file, import FlexmonsterPivot like this:

```
import { FlexmonsterPivot } from './flexmonster.component';
```

Step 5. In the app.module.ts file, declare the FlexmonsterPivot module:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { FlexmonsterPivot } from './flexmonster.component';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent, FlexmonsterPivot
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
}
export class AppModule { }
```

Step 6. Import Flexmonster styles in the src/styles.css file:

```
@import "../node_modules/flexmonster/flexmonster.min.css";
```

After completing these steps, you can use Flexmonster in your Angular 4 application.

The fm-pivot directive and its attributes

The fm-pivot directive embeds the component into the HTML page. The name of each attribute of the fm-pivot directive is surrounded by square brackets. The value of each attribute can be of any type, including an Angular variable available in the current scope, but it must be surrounded by double quotes. Note that **string values must**

be surrounded by single quotes within the double quotes, e.g.
[report]="'https://cdn.flexmonster.com/reports/report.json'".

All attributes are equivalent to those which are passed to the new Flexmonster() API call. Check out the full list of available attributes (/api/new-flexmonster/).

Here is an example demonstrating how different attributes are specified:

```
<h1>Angular 4+/Flexmonster</h1>
<fm-pivot [componentFolder]="'https://cdn.flexmonster.com/'"
          [toolbar]="true"
          [width]="'100%'"
          [height]="500"
          [report]="'https://cdn.flexmonster.com/reports/report.json'"
          (reportcomplete)="onReportComplete()">
    Flexmonster will appear here
</fm-pivot>
```

In the above example notice the following line:

```
(reportcomplete)="onReportComplete()"
```

In Angular, events are specified in round brackets instead of square brackets. This line means that onReportComplete handles the reportcomplete event. Any other event handling can be specified the same way. Here is the full list of Flexmonster events (/api/events/).

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (/doc/customizing-toolbar/)
- How to add localization (/doc/localizing-component/)
- How to customize appearance with CSS (/doc/customizing-appearance/)
- How to define a format for date and time (/doc/date-and-time-formatting/)
- How to set specific options common for all reports (/doc/global-object/)
- How to configure the way that data is exported (/doc/export-and-print/)

3.3. Integration with React

This tutorial will help you integrate Flexmonster with the React framework (https://facebook.github.io/react/).

Prerequisites

To work with React, you will need Node.js and npm. Get it here (https://docs.npmjs.com/getting-started/installing-node) if it’s not already installed on your machine.

One more tool needed is Flexmonster CLI (/doc/cli-overview/), which is the most convenient way to work with Flexmonster. Install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

Now a new flexmonster command is available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

After that, choose one of the following options:

1. Run a sample project from GitHub (#github-sample)
2. Integrate Flexmonster into a React application (#integration)
3. See the examples of Flexmonster usage in React (#examples)
4. Learn how to use Flexmonster with React Hooks (#react-hooks)
5. Improve Flexmonster performance in React (#improve-performance)

Run the sample project from GitHub

Download the sample project with the flexmonster create command. You can choose either the React + ES6 or React + TypeScript project:

React + ES6

```
flexmonster create react es6 -r
```

The flexmonster create react es6 command will do the following:

- Download the .zip archive with the sample React + ES6 project from GitHub.
- Automatically unpack the files in the current folder — as a result, the flexmonster-react-es6-project/ folder will appear in your working directory.

The -r option, which stands for --run, completes these tasks:

- Installs all the npm dependencies described in package.json.
- Compiles the application and runs it in the browser.

React + TypeScript

```
flexmonster create react typescript -r
```

The flexmonster create react typescript command will do the following:

- Download the .zip archive with the sample React + TypeScript project from GitHub.
- Automatically unpack the files in the current folder — as a result, the flexmonster-react-typescript-project/ folder will appear in your working directory.

The -r option, which stands for --run, completes these tasks:

- Installs all the npm dependencies described in package.json.

- Compiles the application and runs it in the browser.

The application can be shut down manually with Ctrl+C.

Integrate Flexmonster into a React application

Follow the steps below to integrate Flexmonster Pivot into a new React application. If you already have the React project, jump to Step 2. Install Flexmonster (#install-flexmonster).

Step 1. Create a new React project

Create a React app by running the commands below in the console. You can choose between React + ES6 and React + TypeScript projects:

React + ES6

```
npx create-react-app my-app
cd my-app
```

React + TypeScript

```
npx create-react-app my-app --typescript
cd my-app
```

Step 2. Install Flexmonster

Install the Flexmonster React module by running this CLI command from the folder containing package.json:

```
flexmonster add react-flexmonster
```

The add command will install the react-flexmonster package to node_modules/ and add it as an npm dependency to package.json.

Step 3. Add Flexmonster styles

Import the styles of Flexmonster.

React + ES6

Add the following import statement to index.js:

```
import 'flexmonster/flexmonster.css';
```

React + TypeScript

Add the following import statement to index.tsx:

```
import 'flexmonster/flexmonster.css';
```

Step 4. Import Flexmonster

Include the FlexmonsterReact component into your React project.

React + ES6

Add the following import statement to App.js:

```
import * as FlexmonsterReact from 'react-flexmonster';
```

React + TypeScript

Add the following import statement to App.tsx:

```
import * as FlexmonsterReact from 'react-flexmonster';
```

Step 5. Create the pivot table

Instructions on how to use Flexmonster in React vary depending on the type of your React project.

React + ES6

Insert a pivot table into App.js:

```
class App extends Component {
  render()
    return (
      <div className="App">
        <FlexmonsterReact.Pivot
         toolbar={true}
         componentFolder="https://cdn.flexmonster.com/"
         width="100%"
         report="https://cdn.flexmonster.com/reports/report.json"
        />
      </div>
    );
  }
}
```

React + TypeScript

Insert a pivot table into App.tsx:

```
class App extends React.Component {
  public render() {
    return (
      <div className="App">
        <FlexmonsterReact.Pivot
         toolbar={true}
         componentFolder="https://cdn.flexmonster.com/"
         width="100%"
         report="https://cdn.flexmonster.com/reports/report.json"
        />
      </div>
    );
  }
}
```

Step 6. Run the project

Run your application from the console:

```
npm start
```

Examples

Both React + ES6 and React + TypeScript sample projects contain several Flexmonster usage examples. You can try them all on the projects’ starting page.

This section gives a detailed description of each example:

- Adding the pivot table (#add-pivot-table)
- Calling events (#use-events)
- Using API calls (#use-methods)
- Updating data (#update-data)
- Customizing the Toolbar (#customize-toolbar)
- Customizing the grid (#customize-grid)
- Integrating with Highcharts (#with-highcharts)
- Integrating with amCharts (#with-amcharts)

Adding the pivot table

The first example demonstrates the basic usage of Flexmonster. See how the toolbar, report, and licenseKey initialization parameters are specified:

- In the React + ES6 project (https://github.com/flexmonster/pivot-react/blob/master/ES6/src/components/ReactFlexmonsterExamples/PivotTableDemo.js)
- In the React + TypeScript project (https://github.com/flexmonster/pivot-react/blob/master/typescript/src/components/ReactFlexmonsterExamples/PivotTableDemo.tsx)

Flexmonster has more initialization parameters. Have a look at all of them (https://www.flexmonster.com/api/new-flexmonster/).

Calling events

This usage example is focused on Flexmonster events. It provides a toggle button for subscribing to all the events and unsubscribing from them. Here is the source code:

- For the React + ES6 project (https://github.com/flexmonster/pivot-react/blob/master/ES6/src/components/ReactFlexmonsterExamples/CallingEvents.js)
- For the React + TypeScript project (https://github.com/flexmonster/pivot-react/blob/master/typescript/src/components/ReactFlexmonsterExamples/CallingEvents.tsx)

When Flexmonster is subscribed to the events, the log output displays:

- Which event was triggered.
- When the event was triggered.
- Details on that event.

See the full list of Flexmonster events in our documentation (https://www.flexmonster.com/api/events/).

Using API calls

The Using API calls section is about customizing the component with API calls. Switch the toggle buttons to:

- Show the pie chart.
- Show the grid.
- Make the component read-only.
- Make the component interactive.

Here is how the functionality is implemented:

- In the React + ES6 project (https://github.com/flexmonster/pivot-react/blob/master/ES6/src/components/ReactFlexmonsterExamples/UsingAPICalls.js)
- In the React + TypeScript project (https://github.com/flexmonster/pivot-react/blob/master/typescript/src/components/ReactFlexmonsterExamples/UsingAPICalls.tsx)

See the full list of Flexmonster API calls (https://www.flexmonster.com/api/methods/).

Updating data

The Updating data section contains an example of data updating at runtime. The example uses the updateData() (https://www.flexmonster.com/api/updatedata/) API call in the function to update the data.

See the source code:

- For the React + ES6 project (https://github.com/flexmonster/pivot-

react/blob/master/ES6/src/components/ReactFlexmonsterExamples/UpdatingData.js)
- For the React + TypeScript project (https://github.com/flexmonster/pivot-react/blob/master/typescript/src/components/ReactFlexmonsterExamples/UpdatingData.tsx)

Customizing the Toolbar

Go to the Customizing the Toolbar section to see the example of Toolbar customization.

We use the beforetoolbarcreated event to invoke the customizeToolbar() function. As a result, a custom tab with a custom functionality is added. See how it is implemented:

- In the React + ES6 project (https://github.com/flexmonster/pivot-react/blob/master/ES6/src/components/ReactFlexmonsterExamples/CustomizingToolbar.js)
- In the React + TypeScript project (https://github.com/flexmonster/pivot-react/blob/master/typescript/src/components/ReactFlexmonsterExamples/CustomizingToolbar.tsx)

Learn more about customizing the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/).

Customizing the grid

The Customizing the grid example demonstrates how the component can be customized.

Switch the toggle buttons to apply or remove customization. Custom grid styles are defined in customizeCellFunction(). Have a look at the source code:

- For the React + ES6 project (https://github.com/flexmonster/pivot-react/blob/master/ES6/src/components/ReactFlexmonsterExamples/CustomizingGrid.js)
- For the React + TypeScript project (https://github.com/flexmonster/pivot-react/blob/master/typescript/src/components/ReactFlexmonsterExamples/CustomizingGrid.tsx)

See our documentation (https://www.flexmonster.com/doc/customizing-grid/) to learn more about cell customizing.

Integrating with Highcharts

See an example of integration with Highcharts in the With Highcharts section. Here is how the integration is implemented:

- In the React + ES6 project (https://github.com/flexmonster/pivot-react/blob/master/ES6/src/components/ReactFlexmonsterExamples/WithHighcharts.js)
- In the React + TypeScript project (https://github.com/flexmonster/pivot-react/blob/master/typescript/src/components/ReactFlexmonsterExamples/WithHighcharts.tsx)

Here are the main elements of this integration:

1. The Highcharts module and Flexmonster Connector for Highcharts.
2. A container for Highcharts.
3. The reportcomplete event; when triggered, the function to draw the chart is invoked.
4. The drawChart() function.

Integration with other charting libraries can be done in a similar way. Check out the integrations we provide (https://www.flexmonster.com/doc/available-tutorials-charts/).

Integrating with amCharts

In the With amCharts section, you can see a dashboard with Flexmonster and amCharts. Here is how the integration is implemented:

- In the React + ES6 project (https://github.com/flexmonster/pivot-react/blob/master/ES6/src/components/ReactFlexmonsterExamples/WithAmcharts.js)
- In the React + TypeScript project (https://github.com/flexmonster/pivot-react/blob/master/typescript/src/components/ReactFlexmonsterExamples/WithAmcharts.tsx)

The key elements of this integration are:

1. The amCharts module and Flexmonster Connector for amCharts (https://www.flexmonster.com/api/all-methods-amcharts/).
2. A container for amCharts.
3. The reportcomplete event; when triggered, the function to draw the chart is invoked.
4. The drawChart() function.

Integration with other charting libraries can be done in a similar way. Check out the integrations we provide (https://www.flexmonster.com/doc/available-tutorials-charts/).

A React/JSX application with Flexmonster

On our GitHub, we also have examples of React/JSX application with Flexmonster Pivot:

- A simple React/JSX app with Flexmonster (https://github.com/flexmonster/pivot-react/blob/master/01-simple-pivot.html)
- A React/JSX app with Flexmonster embedded into an <App/> component (https://github.com/flexmonster/pivot-react/blob/master/02-wrap-with-app.html)
- A React/JSX app with Flexmonster events (https://github.com/flexmonster/pivot-react/blob/master/03-event-handlers.html)

To run these applications, just launch the needed HTML page from a browser.

Using Flexmonster with React Hooks

To use Flexmonster Pivot with React Hooks, import FlexmonsterReact as a class component:

```
import * as FlexmonsterReact from 'react-flexmonster';
```

Now embed Flexmonster into your React Hooks component:

```
import React from 'react';
import * as FlexmonsterReact from 'react-flexmonster';

function PivotTableHooks (props) {
    const ref = React.useRef();

    const onReportComplete = () => {
        console.log(">>>>", ref.current.flexmonster.getReport());
    }

    return <>
        <div className="App">
            <FlexmonsterReact.Pivot
            ref={ref}
            toolbar={true}
            width="100%"
            report="https://cdn.flexmonster.com/reports/report.json"
            reportcomplete={onReportComplete}
          />
        </div>
    </>;
}

export default PivotTableHooks;
```

In the example above, notice this line:

```
console.log(">>>>", ref.current.flexmonster.getReport());
```

It demonstrates how to call Flexmonster's methods when using React Hooks. Here is the full list of Flexmonster API calls (https://www.flexmonster.com/api/methods/).

Improve Flexmonster performance in React

One of the causes of slow Flexmonster performance in React is the babel-loader (https://www.npmjs.com/package/babel-loader) package, which is built in the react-create-app command. By default, babel-loader transpiles files located in the node_modules/ folder, including Flexmonster files.

This issue can be fixed by manually configuring babel-loader, which requires running the npm run eject command. Note that this command's effect is irreversible, so we recommend learning more about npm run eject (https://create-react-app.dev/docs/available-scripts/#npm-run-eject) before running it.

To improve Flexmonster performance, follow the steps below:

Step 1. Run the npm run eject command in the console.

Step 2. Open the webpack.config.js file and exclude Flexmonster from babel-loader:

```
// Process any JS outside of the app with Babel
// Unlike the application JS, we only compile the standard ES features
{
    test: /\.(js|mjs)$/,
    exclude: [
        /node_modules[\\/]flexmonster/,
        /@babel(?:\/|\\{1,2})runtime/
    ],
    loader: require.resolve('babel-loader'),
    …
}
```

If you completed the instructions above, but the performance is still slow, you can try passing data to the component in a different way.

For example, data can be aggregated on a server and passed to Flexmonster in a ready-to-show format. This will reduce the load on the browser and improve performance. This approach is implemented in Flexmonster Data Server (https://www.flexmonster.com/doc/intro-to-flexmonster-data-server/) – our tool for server-side data processing, so you are welcome to use it.

Properties

All available attributes for FlexmonsterReact.Pivot are equivalent to those which are passed to the new Flexmonster() API call. Check out the full list of available attributes (https://www.flexmonster.com/api/new-flexmonster/).

Here is an example demonstrating how different attributes are specified:

```
<FlexmonsterReact.Pivot
 toolbar={true}
 componentFolder="https://cdn.flexmonster.com/"
 width="100%"
 height="600"
 report="https://cdn.flexmonster.com/reports/report.json"
 reportcomplete={this.onReportComplete}
/>
```

In the example above, notice the following line:

```
reportcomplete={this.onReportComplete}
```

This line means that onReportComplete handles the reportcomplete event. Any other event handling can be specified in the same way. Here is the full list of Flexmonster events (https://www.flexmonster.com/api/events/).

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (/doc/customizing-toolbar/)
- How to add localization (/doc/localizing-component/)
- How to customize appearance with CSS (/doc/customizing-appearance/)
- How to define a format for date and time (/doc/date-and-time-formatting/)
- How to set specific options common for all reports (/doc/global-object/)
- How to configure the way that data is exported (/doc/export-and-print/)

3.4. Integration with Vue

This tutorial will help you integrate Flexmonster with the Vue framework (https://vuejs.org/).

Prerequisites

To run a simple application, you will need Node.js and npm. Get it here (https://docs.npmjs.com/downloading-and-installing-node-js-and-npm) if it’s not already installed on your machine.

One more tool needed is Flexmonster CLI (/doc/cli-overview/), which is the most convenient way to work with Flexmonster. Install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

Now a new flexmonster command is available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

After that, choose one of the following options:

1. Run a sample Vue 2 project from GitHub (#github_sample)
2. Integrate Flexmonster into a Vue 2 application (#integration)
3. See the examples of Flexmonster usage in Vue (#examples)
4. Integrate Flexmonster into a Vue 3 application (#integration-vue3)

Run a sample Vue 2 project from GitHub

Create the sample project with the following CLI command:

```
flexmonster create vue -r
```

The flexmonster create vue command will do the following:

- Download the .zip archive with the sample Vue project from GitHub.
- Automatically unpack the files in the current folder — as a result, the flexmonster-vue-project/ folder will appear in your working directory.

The -r option, which stands for --run, completes these tasks:

- Installs all the npm dependencies described in package.json.
- Compiles the application and runs it in the browser.

The application can be shut down manually with Ctrl+C.

Integrate Flexmonster into a Vue 2 application

Follow the steps below to integrate Flexmonster Pivot into a new Vue 2 application. If you already have the Vue project, jump to Step 3. Install Flexmonster. (#install-flexmonster)

To integrate Flexmonster with Vue 3, refer to this section (#integration-vue3).

Step 1. Install the Vue CLI

For this tutorial, you will need the Vue CLI (https://cli.vuejs.org/). If the Vue CLI is not installed, run the following command in the console to install it globally:

```
npm install -g @vue/cli
```

Step 2. Create a Vue project

If you don’t have a Vue CLI app, create it by running these commands in the console:

```
vue create project-name
cd project-name
```

For simple cases, it is enough to select the default configurations when creating a Vue CLI app.

Step 3. Install Flexmonster

Install the Flexmonster Vue module by running this CLI command from the folder containing package.json:

```
flexmonster add vue-flexmonster
```

The add command will install the vue-flexmonster package to node_modules/ and add it as an npm dependency to package.json.

Step 4. Register the vue-flexmonster module

Now the vue-flexmonster module can either be used as a plugin (global registration) or locally registered in your project:

1. To use vue-flexmonster as a plugin, add the following code after existing imports in src/main.js:

   ```
   // Using vue-flexmonster as a plugin (global registration):
   // 1. Importing the vue-flexmonster module and CSS styles
   import Pivot from "vue-flexmonster";
   import 'flexmonster/flexmonster.css';

   // 2. Referring to the vue-flexmonster module as a plugin
   Vue.use(Pivot);
   ```

   Now, any component in your application can use the Pivot.
2. To use vue-flexmonster locally, insert the following code in the <script> section of the component where you need the pivot table (e.g., in the src/App.vue):

   ```
   // Using the vue-flexmonster module (local registration):
   // 1. Importing the vue-flexmonster module and CSS styles
   import {Pivot} from "vue-flexmonster";
   import 'flexmonster/flexmonster.css';

   // 2. Defining the module in the components list
   export default {
       name: 'app',
   ```

```
        components: {
            Pivot
        }
    }
```

Step 5. Add Flexmonster instance to a Vue component

Include the Pivot module in the <template> section of the chosen component (e.g., in the src/App.vue). Make sure the component template contains exactly one root <div> element.

```
<template>
  <div id="app">
    <Pivot
     ref="pivot"
     v-bind:
     report="'https://cdn.flexmonster.com/reports/report.json'">
    </Pivot>
  </div>
</template>
```

Step 6. Run the project

Run your application from the console:

```
npm run serve
```

To see the result, open http://localhost:8080/ in your browser.

The application can be shut down manually with Ctrl+C.

Examples

The sample Vue project with Flexmonster contains several usage examples (https://github.com/flexmonster/pivot-vue/tree/master/src/components/VueFlexmonsterExamples). You can try them all on the project’s starting page.

This section gives a detailed description of each example:

- Adding the pivot table (#add-pivot-table)
- Calling events (#use-events)
- Using API calls (#use-methods)
- Updating data (#update-data)
- Customizing the Toolbar (#customize-toolbar)
- Customizing the grid (#customize-grid)
- Integrating with Highcharts (#with-highcharts)
- Integrating with amCharts (#with-amcharts)

Adding the pivot table

The first example (https://github.com/flexmonster/pivot-vue/blob/master/src/components/VueFlexmonsterExamples/PivotTableDemo.vue#L4) demonstrates the basic usage of Flexmonster. See how the toolbar, report, and licenseKey initialization parameters are specified in Vue.

Flexmonster has more initialization parameters. Have a look at all of them (https://www.flexmonster.com/api/new-flexmonster/).

Calling events

This usage example (https://github.com/flexmonster/pivot-vue/blob/master/src/components/VueFlexmonsterExamples/CallingEvents.vue) is focused on Flexmonster events. It provides a toggle button for subscribing to all the events and unsubscribing from them.

When Flexmonster is subscribed to the events, the log output displays:

- Which event was triggered.
- When the event was triggered.
- Details on that event.

See the full list of Flexmonster events in our documentation (https://www.flexmonster.com/api/events/).

Using API calls

The Using API calls (https://github.com/flexmonster/pivot-vue/blob/master/src/components/VueFlexmonsterExamples/UsingAPICalls.vue) section is about customizing the component with API calls. Switch the toggle buttons to:

- Show the column chart.
- Show the grid.
- Make the component read-only.
- Make the component interactive.

See the full list of Flexmonster API calls (https://www.flexmonster.com/api/methods/).

Updating data

The Updating data (https://github.com/flexmonster/pivot-vue/blob/master/src/components/VueFlexmonsterExamples/UpdatingData.vue) section contains an example of data updating at runtime. The example uses the updateData() (https://www.flexmonster.com/api/updatedata/) API call in the function to update the data.

Customizing the Toolbar

Go to the Customizing the Toolbar (https://github.com/flexmonster/pivot-vue/blob/master/src/components/VueFlexmonsterExamples/CustomizingToolbar.vue) section to see the example of Toolbar customization.

We use the beforetoolbarcreated event to invoke the customizeToolbar() function. As a result, a custom tab with a custom functionality is added.

Learn more about customizing the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/).

Customizing the grid

The Customizing the grid (https://github.com/flexmonster/pivot-vue/blob/master/src/components/VueFlexmonsterExamples/CustomizingGrid.vue) example demonstrates how the component can be customized.

Switch the toggle buttons to apply or remove customization. Custom grid styles are defined in the customizeCellFunction().

See our documentation (https://www.flexmonster.com/doc/customizing-grid/) to learn more about cell customizing.

Integrating with Highcharts

See an example of integration with Highcharts in the With Highcharts (https://github.com/flexmonster/pivot-vue/blob/master/src/components/VueFlexmonsterExamples/WithHighcharts.vue) section.

Here are the main elements of this integration:

1. The Highcharts module and Flexmonster Connector for Highcharts.
2. A container for Highcharts.
3. The reportcomplete event; when triggered, the function to draw the chart is invoked.
4. The drawChart() function.

Integration with other charting libraries can be done in a similar way. Check out the integrations we provide (https://www.flexmonster.com/doc/available-tutorials-charts/).

Integrating with amCharts

In the With amCharts (https://github.com/flexmonster/pivot-vue/blob/master/src/components/VueFlexmonsterExamples/WithAmcharts.vue) section, you can see a dashboard with Flexmonster and amCharts.

The key elements of this integration are:

1. The amCharts module and Flexmonster Connector for amCharts (https://www.flexmonster.com/api/all-methods-amcharts/).
2. A container for amCharts.
3. The reportcomplete event; when triggered, the function to draw the chart is invoked.
4. The drawChart() function.

Integration with other charting libraries can be done in a similar way. Check out the integrations we provide (https://www.flexmonster.com/doc/available-tutorials-charts/).

Integrate Flexmonster into a Vue 3 application

To integrate Flexmonster with Vue 3, we will use the Pivot.vue module from GitHub (https://github.com/flexmonster/vue-flexmonster/blob/master/src/Pivot.vue).

Step 1. If needed, uninstall the vue-flexmonster npm module:

```
npm uninstall vue-flexmonster
```

Step 2. Download the Pivot.vue source code from GitHub (https://github.com/flexmonster/vue-flexmonster/blob/master/src/Pivot.vue) and add it to your project (e.g., to the src/components/ folder).

Step 3. Install the flexmonster npm module:

```
npm install flexmonster
```

Step 4. Import Flexmonster styles in main.js:

```
import { createApp } from 'vue'
import App from './App.vue'

import 'flexmonster/flexmonster.css'

createApp(App).mount('#app')
```

Step 5. In the needed component (e.g., in App.vue), import Flexmonster from Pivot.vue:

```
<script>
  import Pivot from "./components/Pivot"

  export default {
    name: 'App',
    components: {
      Pivot
    }
  }
</script>

<style>...</style>
```

Step 6. Embed Flexmonster into the needed component (e.g., into App.vue):

```
<script>
  import Pivot from "./components/Pivot"

  export default {
    name: 'App',
    components: {
      Pivot
    }
  }
</script>

<template>
  <Pivot
   toolbar
   v-bind:report="'https://cdn.flexmonster.com/reports/report.json'">
  </Pivot>
</template>

<style>...</style>
```

Now you can use Flexmonster in your Vue 3 applications.

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/)
- How to add localization (https://www.flexmonster.com/doc/localizing-component/)
- How to customize appearance with CSS (https://www.flexmonster.com/doc/customizing-appearance/)
- How to define a format for date and time (https://www.flexmonster.com/doc/date-and-time-formatting/)
- How to set specific options common for all reports (https://www.flexmonster.com/doc/global-object/)
- How to configure the way that data is exported (https://www.flexmonster.com/doc/export-and-print/)

3.5. Other integrations

3.5.1. Integration with Python

3.5.1.1. Integration with Django

This tutorial will help you integrate Flexmonster with the Django framework (https://www.djangoproject.com/).

Prerequisites

To run a simple application, you will need Python and Django. Get the latest Python and Django versions (https://docs.djangoproject.com/en/3.0/intro/install/#quick-install-guide) if they’re not already installed on your machine.

After that, choose one of the following options:

1. Run a simple Django and Flexmonster project from GitHub (#github-sample)
2. Integrate Flexmonster into a Django application (#integrate-new)

Run a simple Django and Flexmonster sample from GitHub

Step 1. Download the .zip archive with the sample or clone it from GitHub (https://github.com/flexmonster/pivot-Django) with the following command:

```
git clone https://github.com/flexmonster/pivot-Django pivot-django
cd pivot-django
```

Step 2. Run your application:

```
python manage.py runserver
```

To see the result, open http://localhost:8000/ in your browser.

The application can be shut down manually with Ctrl+C.

Integrate Flexmonster into a Django application

To integrate Flexmonster into a Django app, follow these steps:

Step 1. If you don’t have a Django project, you can create it by running these commands in the console:

```
django-admin startproject pivot_django
cd pivot_django
```

Step 2. In the project, create a new app with the following command:

```
python manage.py startapp pivot_django_app
```

Step 3. Register the app in the project by adding the app’s config function’s name (this can be found in pivot_django_app/apps.py) to the INSTALLED_APPS list in the pivot_django/settings.py file:

```
INSTALLED_APPS = [
     'django.contrib.admin',
     'django.contrib.auth',
     'django.contrib.contenttypes',
     'django.contrib.sessions',
     'django.contrib.messages',
     'django.contrib.staticfiles',
     'pivot_django_app.apps.PivotDjangoAppConfig',
 ]
```

Step 4. Create a folder (e.g., templates/) for HTML templates in the root folder where pivot_django and pivot_django_app are located. Next, create an HTML file (e.g., index.html) and add Flexmonster to it:

```
<h1>Flexmonster integration with Django</h1>
<div id="pivotContainer"></div>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>

<script>
    var pivot = new Flexmonster({
        container: "pivotContainer",
        componentFolder: "https://cdn.flexmonster.com/",
        report: {
            dataSource: {
                type: "json",
                filename: "https://cdn.flexmonster.com/data/data.json"
            }
        }
    });
</script>
```

Step 5. Add the created HTML page to the app’s views. Navigate to pivot_django_app/views.py and insert the following index function there:

```
def index(request):
    return render(request, 'index.html')
```

Step 6. Add the path('', views.index, name='index'), line to the urlpatterns list in pivot_django/urls.py:

```
from pivot_django_app import views
urlpatterns = [
     path('admin/', admin.site.urls),
     path('', views.index, name='index'),
 ]
```

Step 7. Add an os.path.join(BASE_DIR, 'templates') line to the DIRS list in the TEMPLATES list in pivot_django/settings.py:

```
'DIRS': [os.path.join(BASE_DIR, 'templates')],
```

Step 8. Run your application:

```
python manage.py runserver
```

To see the result, open http://localhost:8000/ in your browser.

The application can be shut down manually with Ctrl+C.

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/)
- How to add localization (https://www.flexmonster.com/doc/localizing-component/)
- How to customize appearance with CSS (https://www.flexmonster.com/doc/customizing-appearance/)
- How to define a format for date and time (https://www.flexmonster.com/doc/date-and-time-formatting/)
- How to set specific options common for all reports (https://www.flexmonster.com/doc/global-object/)
- How to configure the way that data is exported (https://www.flexmonster.com/doc/export-and-print/)

3.5.1.2. Integration with Jupyter Notebook

This tutorial will help you integrate Flexmonster with Jupyter Notebook (https://jupyter.org/) applications. Follow these steps to set up a simple project.

Prerequisites

To run a simple application, you will need the Jupyter Notebook. For simplicity, you can use the web version (https://jupyter.org/try) of it, which doesn't require anything to be installed on your computer. You can also follow this guide (https://jupyter.org/install.html) and install the Notebook yourself.

After that, choose one of the following options:

1. Run a simple Jupyter Notebook and Flexmonster sample from GitHub (#github-sample)
2. Integrate Flexmonster into a Jupyter Notebook application (#integrate-new)

Run a simple Jupyter Notebook and Flexmonster sample from GitHub

Step 1. Download the .zip archive with the sample or clone it from GitHub (https://github.com/flexmonster/pivot-jupyter-notebook) with the following command:

```
git clone https://github.com/flexmonster/pivot-jupyter-notebook
```

Step 2a. If using the desktop version of the Jupyter Notebook, run it with the following command:

```
jupyter notebook
```

The Jupyter Notebook will be automatically launched in your browser.

Step 2b. Now, upload the Flexmonster_in_Jupyter_Notebook.ipynb file to the Jupyter Notebook. The file will appear in the Files tab.

Step 3. Run the sample project by selecting the Cell section in the navigation bar and clicking on the Run all option.

Integrate Flexmonster into a Jupyter Notebook application

To integrate Flexmonster into a Jupyter Notebook app, follow these steps:

Step 1. If you don't have an existing Python 3 notebook, open the Jupyter Notebook and create a new Python 3 file.

Step 2. Import the following libraries to work with HTML and JSON in Python, as well as Pandas for data manipulation:

```
from IPython.display import HTML
import json
import pandas as pd
```

Step 3. Define some data with Pandas and convert it to JSON using the orient="records" parameter. This will present the data in the format that Flexmonster requires:

```
data = pd.DataFrame([["Lemon cake", 30, 4.99],["Apple pie", 45, 6.99], ["Raspberry j
am", 70, 3.99]],index=['row 1', 'row 2', 'row 3'], columns=['Product', 'Quantity', '
Price per Item'])
json_data = data.to_json(orient="records")
```

Step 4. Define Flexmonster:

```
flexmonster = {
    "container": "pivotContainer",
    "componentFolder": "https://cdn.flexmonster.com/",
    "report": {
        "dataSource": "json",
        "data": json.loads(json_data)
    }
}
```

Step 5. Convert the flexmonster object to a JSON-formatted string:

```
flexmonster_json_object = json.dumps(flexmonster)
```

Step 6. Define a function to display Flexmonster in HTML (e.g., pivot):

```
def pivot(flexmonster_json_object):
#the format function is needed to insert the Flexmonster object into the script
    code = '''
      <script src="https://cdn.flexmonster.com/flexmonster.js"></script>
      <h1>Flexmonster Integration with the Jupyter Notebook</h1>
      <div id="pivotContainer"></div>
      <script>
       new Flexmonster({});
      </script>
      '''.format(flexmonster_json_object)
      #convert the code string to HTML
    return HTML(code)
```

Step 7. Display the component using the previously defined pivot function:

```
pivot(flexmonster_json_object)
```

Step 8. Run the notebook by selecting the Cell section in the navigation bar and clicking on the Run all option.

The component will appear in an output cell right under the one containing the pivot function call.

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/)
- How to add localization (https://www.flexmonster.com/doc/localizing-component/)
- How to customize the appearance with CSS (https://www.flexmonster.com/doc/customizing-appearance/)
- How to define a format for date and time (https://www.flexmonster.com/doc/date-and-time-formatting/)
- How to set specific options common for all reports (https://www.flexmonster.com/doc/global-object/)
- How to configure the way that data is exported (https://www.flexmonster.com/doc/export-and-print/)

3.5.2. Integration with React Native

This tutorial will help you integrate Flexmonster with the React Native framework (https://facebook.github.io/react-native/). It is based on the React Native: Getting Started (https://facebook.github.io/react-native/docs/getting-started) guide.

Prerequisites

To run a simple application, you will need Node.js and npm. Get it here (https://docs.npmjs.com/downloading-and-installing-node-js-and-npm) if it’s not already installed on your machine.

Then install the Expo CLI (https://expo.io/tools#cli) globally by running:

```
npm install -g expo-cli
```

After that, choose one of the following options:

1. Run the sample project from GitHub (#github-sample)
2. Integrate Flexmonster into a React Native application (#integrate-new)
3. Using methods and events in React Native (#methods-and-events)
4. Customizing Flexmonster in React Native (#customizing)

Run the sample project from GitHub

Step 1. Download the .zip archive with the sample project or clone it from GitHub (https://github.com/flexmonster/pivot-react-native) with the following command:

```
git clone https://github.com/flexmonster/pivot-react-native
cd pivot-react-native
```

Step 2. Install the npm packages described in package.json:

```
npm install
```

Step 3. Run your application from the console:

```
expo start
```

The application can be shut down manually with Ctrl+C.

The sample project contains several examples of using Flexmonster methods and events in React Native.

When initialized, the component is subscribed to the cellclick event (https://www.flexmonster.com/api/cellclick/). As a cell is clicked for the first time, the cellclick's handler is removed, and the component subscribes to the update event (https://www.flexmonster.com/api/update/).

The application also has two buttons: Show chart and Show grid. The Show chart button switches to the charts view using the showCharts() method (https://www.flexmonster.com/api/showcharts/). Show grid uses the showGrid() method (https://www.flexmonster.com/api/showgrid/) to switch to the grid view.

Learn more about using methods and events in this section (#methods-and-events).

Integrate Flexmonster into a React Native application

To integrate Flexmonster into a React Native app, follow these steps:

Step 1. If you don't already have a React Native app, create one by running these commands in the console:

```
expo init my-app --template blank
cd my-app
```

Step 2. If you created a new app, install the packages described in package.json:

```
npm install
```

Step 3. Install the Flexmonster React Native module with the following command:

```
npm install react-native-flexmonster --save
```

Step 4. Import FlexmonsterReactNative into App.js:

```
import * as FlexmonsterReactNative from 'react-native-flexmonster';
```

Step 5. Insert Flexmonster Pivot into App.js:

```
export default function App() {
    return (
        <View style={{ flex: 1 }}>
            <FlexmonsterReactNative.Pivot
             report="https://cdn.flexmonster.com/reports/report.json"
            />
        </View>
     );
}
```

Step 6. Run your application from the console:

```
expo start
```

The application can be shut down manually with Ctrl+C.

Using methods and events in React Native

The React Native module provides a lot of Flexmonster's methods and events that make the component highly customizable. See which methods (https://github.com/flexmonster/react-native-flexmonster/blob/master/src/index.js#L17-L187) and events (https://github.com/flexmonster/react-native-flexmonster/blob/master/src/index.js#L288-L328) are available out of the box.

This section explains how to use events and methods in React Native.

Create a React reference

To use Flexmonster methods and events in React Native, we will need a React ref (https://reactjs.org/docs/refs-and-the-dom.html) to the Flexmonster instance. Create a ref and attach it to the FlexmonsterReactNative.Pivot element:

```
this.flexmonster = React.createRef();

<FlexmonsterReactNative.Pivot
 ref={(flexmonster) => { this.flexmonster = flexmonster }}
 report="https://cdn.flexmonster.com/reports/report.json"
/>
```

Now we can refer to the Flexmonster instance throughout the React component.

See how the ref is attached to the component in the sample project (https://github.com/flexmonster/pivot-react-native/blob/master/components/PivotTable.js#L86-L88).

Use methods

A method’s usage depends on its role. Methods can be of two role types:

- Methods that perform an action (#action)
- Methods that return a value (#value)

Methods that perform an action

To call methods that perform some action, just use the ref to Flexmonster instance (#create-ref):

```
this.flexmonster = React.createRef();

function showMyChart() {
    this.flexmonster.showCharts("pie");
}

return (
    <View style={{ flex: 1 }}>
      <FlexmonsterReactNative.Pivot
       ref={(flexmonster) => { this.flexmonster = flexmonster }}
       report="https://cdn.flexmonster.com/reports/report.json"
      />
    </View>
);
```

See how methods that perform an action are used in our sample project (https://github.com/flexmonster/pivot-react-native/blob/master/components/PivotTable.js#L54-L66).

Methods that return a value

Methods that return a value are called similar to methods that perform an action. To use their result, catch it with the then() method (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then):

```
this.flexmonster = React.createRef();

function reportComplete() {
    this.flexmonster.getReport().then(function(report) {
        console.log(report);
    });
}


return (
    <View style={{ flex: 1 }}>
      <FlexmonsterReactNative.Pivot
       ref={(flexmonster) => { this.flexmonster = flexmonster }}
       report="https://cdn.flexmonster.com/reports/report.json"
      />
    </View>
);
```

Use events

You can subscribe to events in two ways:

- Via props (#via-props)
- Via the on() method (#via-on)

Subscribing to events via props

To subscribe to events via props, just define the needed event as a FlexmonsterReactNative.Pivot prop and assign an event handler to it:

```
return (
    <View style={{ flex: 1 }}>
      <FlexmonsterReactNative.Pivot
       cellclick={this.onclickHandler}
       report="https://cdn.flexmonster.com/reports/report.json"
      />
    </View>
);

onclickHandler = () => {
    alert("The cell was clicked!");
}
```

The sample React Native project (https://github.com/flexmonster/pivot-react-native/blob/master/components/PivotTable.js#L85) demonstrates how to subscribe to an event via props.

Subscribing to events via the on() method

You can also subscribe to an event using the on() API call (https://www.flexmonster.com/api/on/). Now we will need the previously created this.flexmonster ref (#create-ref):

```
return (
    <View style={{ flex: 1 }}>
      <FlexmonsterReactNative.Pivot
       ref={(flexmonster) => { this.flexmonster = flexmonster }}
       report="https://cdn.flexmonster.com/reports/report.json"
      />
    </View>
);

myFunction = () => {
    this.flexmonster.on("aftergriddraw", () => {
        alert("aftergriddraw");
    });
}
```

To unsubscribe from an event, use the off() method (https://www.flexmonster.com/api/off/):

```
return (
    <View style={{ flex: 1 }}>
      <FlexmonsterReactNative.Pivot
       ref={(flexmonster) => { this.flexmonster = flexmonster }}
       report="https://cdn.flexmonster.com/reports/report.json"
      />
    </View>
);

myFunction = () => {
    this.flexmonster.on("aftergriddraw", () => {
        alert("aftergriddraw");
        this.flexmonster.off("aftergriddraw");
    });
}
```

Have a look at the sample React Native project (https://github.com/flexmonster/pivot-react-native/blob/master/components/PivotTable.js#L48-L51) and see how on() and off() methods are used.

Customizing Flexmonster in React Native

This section describes the specifics of customizing Flexmonster in React Native.

The Flexmonster React Native module embeds the component into a React Native application using the WebView (https://www.npmjs.com/package/react-native-webview) library, which creates a separate browser window with Flexmonster. Since WebView is an independent part of the application, the component cannot be accessed directly from the app and vice versa.

This approach comes with some limitations. Methods that are defined in the application and require direct access to Flexmonster cannot be used. This includes, for example, customizeCell, customizeContextMenu, and beforetoolbarcreated.

To use these features, you need to modify the Flexmonster React Native module. Follow the steps below to see how the React Native module can be customized.

Step 1. Download the sample project from GitHub

Follow steps 1-3 from the guide on how to run the sample project from GitHub (#github-sample).

Step 2. Download the Flexmonster React Native module from GitHub

The sample project includes the Flexmonster React Native module as an npm dependency, but we will connect it manually to add custom functionality.

Download the .zip archive with the module from GitHub (https://github.com/flexmonster/react-native-flexmonster) or run the following command in the console:

```
git clone https://github.com/flexmonster/react-native-flexmonster.git
```

Step 3. Add the module to your project

Copy the src/index.js file from the react-native-flexmonster/ folder to the project folder (e.g., pivot-react-native/). Rename the file if needed (e.g., to react-native-flexmonster.js).

Step 4. Use the module from GitHub

Replace the React Native module from npm with the module from GitHub by updating the module import statement in the App.js file:

```
import * as FlexmonsterReactNative from './react-native-flexmonster';
```

Step 5. Customize the module

Make the necessary updates to the react-native-flexmonster.js file. Note that all the updates should be made in the HTML template (https://github.com/flexmonster/react-native-flexmonster/blob/master/src/index.js#L336).

The steps below describe how to customize cells on the grid:

1. Add a customizeCellFunction to the module’s htmlTemplate variable:

   ```
   <script>
       new Flexmonster({
           // initialization parameters
       });
       function customizeCellFunction (cell, data) {
           if (data.type == "value") {
               if (data.rowIndex % 2 == 0) {
                   cell.addClass("alter1");
               } else {
                   cell.addClass("alter2");
               }
           }
       }
       ${this.registerEvents()
   </script>
   ```

2. Add CSS classes that will be applied to the rows. This can be done right after the <script></script> section of the htmlTemplate variable:

   ```
   <script>
       new Flexmonster(
           // initialization parameters
       );
   ```

```
    function customizeCellFunction (cell, data) {
        // function implementation
    }
    ${this.registerEvents()}
</script>
<style>
    #fm-pivot-view #fm-grid-view div.alter1 {
        background-color: #f7f7f7;
    }
    #fm-pivot-view #fm-grid-view div.alter2 {
        background-color: #fcfcfc;
    }
</style>
```

3. Call the customizeCell method and pass the customizeCellFunction to it. Note that customizeCell can be defined in two ways:

As a regular API call

```
<script>
    new Flexmonster({
        // initialization parameters
    });
    function customizeCellFunction (cell, data) {
        // function implementation
    }
    ${this.registerEvents()}
    flexmonster.customizeCell(customizeCellFunction);
</script>
```

As an initialization parameter

```
new Flexmonster({
    container: "pivotContainer",
    //other initialization parameters
    report: JSON.parse('${JSON.stringify(this.props.report)}'),
    customizeCell: customizeCellFunction
});
```

Step 6. Run the project

Run the project from the console with the following command:

```
expo start
```

Now all the cells on the grid will be customized by the customizeCellFunction. Other customizations can be achieved in the same way.

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/)
- How to add localization (https://www.flexmonster.com/doc/localizing-component/)
- How to customize appearance with CSS (https://www.flexmonster.com/doc/customizing-appearance/)
- How to define a format for date and time (https://www.flexmonster.com/doc/date-and-time-formatting/)
- How to set specific options common for all reports (https://www.flexmonster.com/doc/global-object/)
- How to configure the way that data is exported (https://www.flexmonster.com/doc/export-and-print/)

3.5.3. Integration with AngularJS (v1.x)

This tutorial will help you integrate the pivot table with the AngularJS framework (https://angularjs.org/). To integrate Flexmonster with the Angular framework, see integration with Angular (https://www.flexmonster.com/doc/integration-with-angular/).

Follow these steps to set up a simple AngularJS project:

Step 1. Create a new AngularJS project

1. Create a new folder for the project, e.g. my-angular-project/.
2. Create an index.html file in the my-angular-project/ folder with a simple AngularJS app inside:

```
<!DOCTYPE html>
<html ng-app="App">
  <head>
    <title>My AngularJS/Flexmonster Project</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.js"><
/script>
    <script type="text/javascript">angular.module("App", [ ]);</script>
  </head>
  <body>
  </body>
</html>
```

Step 2. Add Flexmonster dependencies

Add the Flexmonster library to index.html:

```
<!DOCTYPE html>
<html ng-app="App">
  <head>
    <title>My AngularJS/Flexmonster Project</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.js"><
/script>
    <script type="text/javascript">angular.module("App", [ ]);</script>
    <script src="https://cdn.flexmonster.com/flexmonster.js"></script>
  </head>
  <body>
  </body>
</html>
```

Step 3. Initialize the pivot table

Add the flexmonster module to the App and use the fm-pivot directive to add the pivot table to index.html:

```
<!DOCTYPE html>
<html ng-app="App">
  <head>
    <title>My AngularJS/Flexmonster Project</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.js"><
/script>
    <script type="text/javascript">angular.module("App", ["flexmonster"]);</script>
    <script src="https://cdn.flexmonster.com/flexmonster.js"></script>
  </head>
  <body>
    <div fm-pivot
         fm-component-folder="https://cdn.flexmonster.com/"
         fm-toolbar="true">
    </div>
  </body>
</html>
```

Step 4. Load a sample report

To see some data on the grid add the fm-report attribute with the report’s URL:

```
<!DOCTYPE html>
<html ng-app="App">
  <head>
    <title>My AngularJS/Flexmonster Project</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.js"><
/script>
    <script type="text/javascript">angular.module("App", ["flexmonster"]);</script>
    <script src="https://cdn.flexmonster.com/flexmonster.js"></script>
  </head>
  <body>
    <div fm-pivot
         fm-component-folder="https://cdn.flexmonster.com/"
         fm-toolbar="true"
         fm-report="'https://cdn.flexmonster.com/reports/report.json'">
    </div>
  </body>
</html>
```

The fm-pivot directive and its attributes

The fm-pivot directive embeds the component into the HTML page. The value of each attribute can be of any type, including an Angular variable available in the current scope, but it must be surrounded by double quotes. Note that **string values must be surrounded by single quotes within the double quotes**, e.g. fm-report="'https://cdn.flexmonster.com/reports/report.json'".

All attributes are equivalent to those that are passed to the new Flexmonster() API call. The only difference is that fm- prefix is added before each of them. For example, use fm-report instead or report and fm-component-folder instead of componentFolder. Check out the full list of available attributes here (/api/new-flexmonster/).

Examples

Find more examples of integrating the pivot table with the AngularJS framework on GitHub (https://github.com/flexmonster/pivot-angularjs).

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (/doc/customizing-toolbar/)
- How to add localization (/doc/localizing-component/)
- How to customize appearance with CSS (/doc/customizing-appearance/)
- How to define a format for date and time (/doc/date-and-time-formatting/)
- How to set specific options common for all reports (/doc/global-object/)
- How to configure the way that data is exported (/doc/export-and-print/)

3.5.4. Integration with TypeScript

This tutorial will help you integrate Flexmonster with TypeScript (https://www.typescriptlang.org/).

Prerequisites

The most convenient way to work with Flexmonster Pivot is Flexmonster CLI (/doc/cli-overview/) — a command-line interface tool for Flexmonster. Install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

Now a new flexmonster command is available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

Then choose one of the following approaches:

1. A TypeScript application with Flexmonster Pivot (#typescript)
2. A TypeScript + Webpack application with Flexmonster Pivot (#webpack)
3. TypeScript type definitions for Flexmonster (#type-definitions)

A TypeScript application with Flexmonster Pivot

Create the sample project with the following CLI command:

```
flexmonster create typescript -r
```

The flexmonster create typescript command will do the following:

- Download the .zip archive with the sample TypeScript project from GitHub.
- Automatically unpack the files in the current folder — as a result, the flexmonster-typescript-project/ folder will appear in your working directory.

The -r option, which stands for --run, completes these tasks:

- Installs all the npm dependencies described in package.json.
- Launches the index.html file from a browser.

When the content of main.ts is changed, use the following command to recompile main.js:

```
tsc main.ts
```

A TypeScript + Webpack application with Flexmonster Pivot

Create the sample project with the following CLI command:

```
flexmonster create typescript webpack -r
```

The flexmonster create typescript webpack command will do the following:

- Download the .zip archive with the sample TypeScript + Webpack project from GitHub.
- Automatically unpack the files in the current folder — as a result, the flexmonster-typescript-webpack-project/ folder will appear in your working directory.

The -r option, which stands for --run, completes these tasks:

- Installs all the npm dependencies described in package.json.
- Compiles the application and runs it in the browser.

TypeScript type definitions for Flexmonster

TypeScript type definitions for Flexmonster can be found inside the node_modules/flexmonster/types/flexmonster.d.ts file.

To install the flexmonster npm package to node_modules/, run the following command from the folder containing package.json:

```
flexmonster add flexmonster
```

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/)
- How to add localization (https://www.flexmonster.com/doc/localizing-component/)
- How to customize appearance with CSS (https://www.flexmonster.com/doc/customizing-appearance/)
- How to define a format for date and time (https://www.flexmonster.com/doc/date-and-time-formatting/)
- How to set specific options common for all reports (https://www.flexmonster.com/doc/global-object/)
- How to configure the way that data is exported (https://www.flexmonster.com/doc/export-and-print/)

3.5.5. Integration with R Shiny

This tutorial will help you integrate Flexmonster with the R Shiny application. It is based on the how to build a Shiny app (https://shiny.rstudio.com/articles/build.html) guide.

Prerequisites

To run a simple application, you will need R. Get it here (https://www.r-project.org/) if it’s not already installed on your machine.

Then open the R console and install the Shiny package (https://rstudio.com/products/shiny/) by running:

```
install.packages("shiny")
```

After that, choose one of the following options:

1. Run a simple R Shiny and Flexmonster sample from GitHub (#github-sample)
2. Integrate Flexmonster into a Shiny application (#integrate-new)

Run a simple R Shiny and Flexmonster sample from GitHub

Step 1. Download the .zip archive with the sample or clone it from GitHub (https://github.com/flexmonster/pivot-r-shiny) with the following command:

```
git clone https://github.com/flexmonster/pivot-r-shiny my-proj
cd my-proj
```

Step 2. Run your application from the R console:

```
library(shiny)
runApp('~/my-proj')
```

The application will be automatically opened in your browser.

It can be shut down manually with Ctrl+C.

If any errors appear on this step, refer to the troubleshooting subsection (https://www.flexmonster.com/console/post.php?post=25135&action=edit#troubleshooting).

Integrate Flexmonster into a Shiny application

To integrate Flexmonster into an R Shiny application, follow these steps:

Step 1. If you don’t have an R Shiny app, create a new empty folder (e.g., shinyapp/) with an empty app.r file inside.

Step 2. In the app.r file, import Shiny into your application:

```
library(shiny)
```

Step 3. In the app.r file, define a simple server function:

```
server <- function(input, output) {

}
```

Step 4. In an .html file (e.g., index.html), create a <div> container for the component:

```
<div id="pivotContainer">The component will appear here</div>
```

Step 5. Include flexmonster.js in your HTML page:

```
<div id="pivotContainer">The component will appear here</div>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>
```

Step 6. Add a simple script to embed the component:

```
<div id="pivotContainer">The component will appear here</div>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>
<script>
    var pivot = new Flexmonster({
          container: "pivotContainer",
          componentFolder: "https://cdn.flexmonster.com/",
          toolbar: true
    });
</script>
```

Step 7. Configure a simple report:

```
 <div id="pivotContainer">The component will appear here</div>
 <script src="https://cdn.flexmonster.com/flexmonster.js"></script>
  <script type="text/javascript">
      var pivot = new Flexmonster({
          container: "pivotContainer",
          componentFolder: "https://cdn.flexmonster.com/",
          toolbar: true,
          report: {
              dataSource: {
                  type: "json",
                  filename: "data/data.json"
              }
          }
      });
 </script>
```

Step 8. In the app.r file, define the user interface by calling the htmlTemplate function with the HTML file path (e.g., index.html) as a parameter.

```
ui <- htmlTemplate("index.html")
```

Step 9. Add the shinyApp function that uses the ui object and the server function to build a Shiny app object:

```
shinyApp(ui, server)
```

Step 10. Run your application from the R console:

```
library(shiny)
runApp('~/shinyapp')
```

The application will be automatically opened in your browser.

It can be shut down manually with Ctrl+C.

If any errors appear on this step, refer to the troubleshooting subsection (#troubleshooting).

Troubleshooting

While completing this tutorial, users may face problems with running the app, especially macOS or Linux users. This subsection provides simple instructions on how to fix errors that may be encountered.

Error: No Shiny application exists at the path

If you followed the tutorial accurately and got this error, try the following:

- Make sure the right path to the project folder was specified when running the app. Note that if the app is run right in the project folder (e.g., my-proj/), the runApp function should be called without parameters:

```
library(shiny)
runApp()
```

The following instructions only apply to UNIX-like systems (macOS, Linux, etc.):

- Try starting the R interactive shell without superuser privileges. It can be done with the following command:

  ```
  R
  ```

  Then run the app with commands from the guide:

  ```
  library(shiny)
  runApp('~/my-proj')
  ```

- Running the app right from the terminal may also help. To do this, open the terminal in the project folder (e.g., my-proj/) and run the following command:

  ```
  R -e "shiny::runApp()"
  ```

- The following command can also be used to run the app from the terminal opened in the project folder:

  ```
  Rscript app.R
  ```

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/)
- How to add localization (https://www.flexmonster.com/doc/localizing-component/)
- How to customize the appearance with CSS (https://www.flexmonster.com/doc/customizing-appearance/)
- How to define a format for date and time (https://www.flexmonster.com/doc/date-and-time-formatting/)
- How to set specific options common for all reports (https://www.flexmonster.com/doc/global-object/)
- How to configure the way that data is exported (https://www.flexmonster.com/doc/export-and-print/)

3.5.6. Integration with jQuery

This tutorial will help you integrate the pivot table with the jQuery library (https://jquery.com/). We provide the following options:

1. Run the sample project from GitHub (#sample-project)
2. Integrate Flexmonster into a jQuery application (#new-project)

Run the sample project from GitHub

Step 1. Download the project

Download the .zip archive with the sample project or clone it from GitHub (https://github.com/flexmonster/pivot-jquery)with the following command:

```
git clone https://github.com/flexmonster/pivot-jquery
```

Step 2. See the results

Open the pivot-jquery/index.html file in the browser to see the result.

Integrate Flexmonster into a jQuery application

If you already have a project with jQuery, jump to Step 2. Add Flexmonster (#add-flexmonster). Otherwise, follow the steps below to set up a simple project:

Step 1. Create a new jQuery project

1. Create a new folder for the project, e.g. my-jquery-project/.
2. Create an index.html file inside my-jquery-project/ and include jQuery:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My jQuery/Flexmonster Project</title>
  </head>
  <body>
    <script src="https://cdn.flexmonster.com/lib/jquery.min.js"></script>
  </body>
</html>
```

Step 2. Add Flexmonster

Add the Flexmonster library to index.html:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My jQuery/Flexmonster Project</title>
  </head>
  <body>
    <script src="https://cdn.flexmonster.com/lib/jquery.min.js"></script>
    <script src="https://cdn.flexmonster.com/flexmonster.js"></script>
  </body>
</html>
```

Step 3. Add a container for Flexmonster

Create a <div> container for the component:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My jQuery/Flexmonster Project</title>
  </head>
  <body>
    <script src="https://cdn.flexmonster.com/lib/jquery.min.js"></script>
    <script src="https://cdn.flexmonster.com/flexmonster.js (https://cdn.flexmonster
.com/flexmonster.js)"></script>

    <div id="pivotContainer">The component will appear here</div>
  </body>
</html>
```

Step 4. Initialize the pivot table

Add a simple script to embed the component:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My jQuery/Flexmonster Project</title>
  </head>
  <body>
    <script src="https://cdn.flexmonster.com/lib/jquery.min.js"></script>
    <script src="https://cdn.flexmonster.com/flexmonster.js"></script>

    <div id="pivotContainer">The component will appear here</div>
    <script>
      var pivot = $("#pivotContainer").flexmonster({
          componentFolder: "https://cdn.flexmonster.com/",
          toolbar: true
      });
    </script>
  </body>
</html>
```

Step 5. Load a sample report

To see some data on the grid, add the report attribute with the report URL:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My jQuery/Flexmonster Project</title>
  </head>
  <body>
    <script src="https://cdn.flexmonster.com/lib/jquery.min.js"></script>
    <script src="https://cdn.flexmonster.com/flexmonster.js"></script>

    <div id="pivotContainer">The component will appear here</div>
    <script>
      var pivot = $("#pivotContainer").flexmonster({
          componentFolder: "https://cdn.flexmonster.com/",
          toolbar: true,
          report: "https://cdn.flexmonster.com/reports/report.json"
      });
    </script>
  </body>
</html>
```

Open the index.htmlfile in the browser — the component with a sample report is embedded into your project.

Check out a sample on JSFiddle (https://jsfiddle.net/flexmonster/u6yg86Lz/). If you have any problems — visit our troubleshooting section (#troubleshooting).

Initial jQuery call to embed the component

$("#pivotContainer").flexmonster({
componentFolder: String,
global: Report Object (/api/report-object/),
width: Number | String,
height: Number | String,
report: Report Object (/api/report-object/) | String,
toolbar: Boolean,
customizeCell: Function,
licenseKey: String
})

[starting from version 2.3]

Embeds the component into the HTML page.
As a parameter the jQuery call gets #pivotContainer – this is the id of the HTML element that you want to have as the container for the component.
This method allows you to insert the component into your HTML page and to provide it with all the necessary information for the initialization.

Starting from version 2.3 you can preset options for all reports using the global object.

Note: do not forget to import jQuery and flexmonster.js before you start working with it.

Parameters

- componentFolder – String. The URL of the component's folder that contains all the necessary files. It is also used as the base URL for report files, localization files, styles, and images. *Default value*: *"flexmonster/"*.
- global – Report Object (/api/report-object/). Allows you to preset options for all reports. These options can be overridden for specific reports. This object's structure is the same as for report.
- width – Number | String. The width of the component on the page (either in pixels or as a percentage). *Default value*: *"100%"*.
- height – Number | String. The height of the component on the page (either in pixels or as a percentage). *Default value: 500*.
- report – Report Object (/api/report-object/) | String. The property to set a report. It can be an inline report JSON or a URL to the report JSON. XML reports are also supported for backward compatibility.
- toolbar – Boolean. The parameter to embed the Toolbar (true) or not (false). *Default value: false*.

- customizeCell – Function. Allows the customization of separate cells. For more info, take a look at customizeCell definition and examples (https://www.flexmonster.com/api/customizecell/).
- customizeContextMenu – Function. Allows the customization of the context menu. For more info, take a look at customizeContextMenu definition and examples (https://www.flexmonster.com/api/customizecontextmenu/).
- licenseKey – String. The license key.

Event handlers can also be set as properties for the jQuery call. Check out the list here (/api/events/).

All of the parameters are optional. If you run $("#pivotContainer").flexmonster() – an empty component without the Toolbar will be added with the default width and height.

Returns

Object – the reference to the embedded pivot table. If you want to work with multiple instances on the same page use these objects. All API calls are available through them.

After initialization, you can obtain an instance reference to the created component by using this selector:
var pivot = $("#pivot").data("flexmonster");

Installation troubleshooting

If you are facing any problems embedding the component, go to the page where the component should be displayed and open the browser console to check for any errors. Try the following steps:

- If you get the alert "Flexmonster: jQuery is not loaded.", check the path to jQuery in your HTML page.
- If the page is blank and you see the following error in the console: "Uncaught TypeError: $(...).flexmonster is not a function", check the path to Flexmonster in your HTML page.
- If you get the alert "Unable to load dependencies. Please use 'componentFolder' parameter to set the path to 'flexmonster' folder.", add the componentFolder parameter to your $.flexmonster() call (e.g., "https://cdn.flexmonster.com/"):

```
<div id="pivotContainer">The component will appear here</div>
<script src="https://cdn.flexmonster.com/lib/jquery.min.js"></script>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>

<script>
  var pivot = $("#pivotContainer").flexmonster({
      toolbar: true,
      // set the appropriate path
      componentFolder: "https://cdn.flexmonster.com/",
  });
</script>
```

- If you get the error "Uncaught TypeError: Cannot set property '_generatePosition' of undefined" or something similar, try updating the jQuery UI. The minimum recommended version is 1.9.2.
- If you get the error "Uncaught TypeError: $.parseXML is not a function" or something similar, try updating jQuery. The minimum recommended version is 1.7.

What’s next?

- How to customize the Toolbar (/doc/customizing-toolbar/)
- How to add localization (/doc/localizing-component/)
- How to customize appearance with CSS (/doc/customizing-appearance/)
- How to define a format for date and time (/doc/date-and-time-formatting/)

- How to set specific options common for all reports (/doc/global-object/)
- How to configure the way that data is exported (/doc/export-and-print/)

3.5.7. Integration with Ionic

This guide will help you integrate Flexmonster with the Ionic framework (https://ionicframework.com/), which allows creating cross-platform mobile applications.

Prerequisites

To run a sample application, you will need Node.js and npm. Get it here (https://docs.npmjs.com/getting-started/installing-node) if it’s not already installed on your machine.

After that, choose one of the following options:

- Run the sample project from GitHub (#sample-project)
- Integrate Flexmonster into an Ionic application (#integration)

Run the sample project from GitHub

Follow the steps below to run the sample application demonstrating Flexmonster integration with either Ionic 5 + React or Ionic 3 + Angular.

Step 1. Download or clone the sample project

Download the .zip archive with the sample project or clone it with the following command:

Ionic React project

```
git clone https://github.com/flexmonster/pivot-ionic-react.git pivot-ionic
cd pivot-ionic
```

Ionic Angular project

```
git clone https://github.com/flexmonster/pivot-ionic.git pivot-ionic
cd pivot-ionic
```

Step 2. Install npm packages

Install npm packages described in the package.json file with the following command:

```
npm install
```

Step 3. Run the project

Run the sample application with the following command:

```
npm run start
```

Integrate Flexmonster into an Ionic application

Steps to embed Flexmonster into an application vary depending on the Ionic application’s type.

Ionic React project

Flexmonster can be integrated into an Ionic React application in the same way it is integrated into React+Typescript application. See integration with React+Typescript (/doc/integration-with-react/#!typescript) for details.

To create a new Ionic React application, refer to the Ionic documentation (https://ionicframework.com/docs/react/quickstart).

Ionic Angular project

Flexmonster can be integrated into an Ionic Angular application in the same way it is integrated into Angular application. See integration with Angular (/doc/integration-with-angular/) for details.

To create a new Ionic Angular application, refer to the Ionic documentation (https://ionicframework.com/docs/angular/overview).

Flexmonster attributes

The usage of Flexmonster attributes depends on the Ionic application’s type.

Ionic React project

In the application designed using Ionic React (https://ionicframework.com/docs/react), the FlexmonsterReact.Pivot component’s usage is the same as in React.

All attributes are equivalent to those passed to the new Flexmonster() API call. Check out the full list of available attributes (/api/new-flexmonster/).

Here is an example demonstrating how different attributes are specified:

```
<FlexmonsterReact.Pivot
 toolbar = {true}
 report = "https://cdn.flexmonster.com/reports/report.json"
 width = "100%"
 reportcomplete = {this.reportCompleteHandler}
/>
```

In the above example, notice the following line:

```
reportcomplete = {this.reportCompleteHandler}
```

Here, we add reportCompleteHandler to the reportcomplete event. Event handlers can be specified for all available Flexmonster events. Here is the full list of Flexmonster events (/api/events/).

Ionic Angular project

In the application designed using Ionic Angular (https://ionicframework.com/docs/angular/overview), the fm-pivot directive's usage is the same as in Angular: the name of each directive's attribute is surrounded by square brackets, and each attribute's value can be of any type, including an Angular variable.

To use a string value, enclose it by single quotes within the double quotes (e.g., [report]="'https://cdn.flexmonster.com/reports/report.json'").

All attributes are equivalent to those passed to the new Flexmonster() API call. Check out the full list of available attributes (/api/new-flexmonster/).

Here is an example demonstrating how different attributes are specified:

```
<fm-pivot
    [toolbar]="true"
    [width]="'100%'"
    [height]="500"
    [report]="'https://cdn.flexmonster.com/reports/report.json'"
    (reportcomplete)="onReportComplete()">
   Flexmonster will appear here
</fm-pivot>
```

In the above example, notice the following line:

```
(reportcomplete)="onReportComplete()"
```

Here, we add the onReportComplete handler to the reportcomplete event. In Angular, events are surrounded by round brackets instead of square brackets. Here is the full list of Flexmonster events (/api/events/).

What's next?

You may be interested in the following articles:

- How to customize the Toolbar (/doc/customizing-toolbar/)
- How to add localization (/doc/localizing-component/)

- How to customize the appearance with CSS (/doc/customizing-appearance/)
- How to configure the way that data is exported (/doc/export-and-print/)

3.5.8. Integration with Electron.js

This tutorial describes how to integrate Flexmonster Pivot with Electron.js (https://www.electronjs.org/) – a JavaScript framework that allows creating cross-platform desktop applications.

Prerequisites

To work with Electron.js, you need Node.js and npm. Get it here (https://docs.npmjs.com/downloading-and-installing-node-js-and-npm) if it’s not already installed on your machine.

One more tool needed is Flexmonster CLI (/doc/cli-overview/), which is the most convenient way to work with Flexmonster. Install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

Now a new flexmonster command is available in the console. Learn more about Flexmonster CLI and its commands in our documentation (/doc/cli-overview/).

After that, choose one of the following options:

- Run the sample project from GitHub (#github-sample)
- Integrate Flexmonster into an Electron application (#integration)

Run the sample project from GitHub

Create the sample project with the following CLI command:

```
flexmonster create electron -r
```

The flexmonster create electron command will do the following:

- Download the .zip archive with the sample Electron project from GitHub.
- Automatically unpack the files in the current folder — as a result, the flexmonster-electron-project/ folder will appear in your working directory.

The -r option, which stands for --run, completes these tasks:

- Installs all the npm dependencies described in package.json.
- Compiles the application and runs it.

As a result, a desktop application with the pivot table embedded will be launched.

Integrate Flexmonster into an Electron application

This guide is based on the first Electron app tutorial (https://www.electronjs.org/docs/tutorial/first-app). Follow the steps below to integrate Flexmonster Pivot into a new Electron.js application. If you already have the Electron project, jump to Step 6. Install Flexmonster (#install-flexmonster).

Step 1. Initialize package.json file

Create an empty folder for the new Electron application (e.g., pivot-electron) and run the following command from that folder:

```
npm init
```

npm will start creating a basic package.json file. When initializing package.json, notice the "main" field – a script specified in this field is the application's entry point. In this tutorial, main.js is the entry point.

As a result, the package.json file should look similar to the following:

```
{
   "name": "pivot-electron",
   "version": "1.0.0",
   "main": "main.js"
}
```

If there is no "main" field specified in package.json, the default entry point is index.js.

Step 2. Add a script to run the Electron app

By default, the npm start command runs the application with Node.js. To run the application with Electron, add the following start script to package.json:

```
"scripts": {
    "start": "electron .",
},
```

Step 3. Install Electron.js

Install Electron.js in the project by running the following command in the console:

```
npm install electron
```

Step 4. Create the HTML file

HTML files define the Electron application's user interface. Create the HTML file (e.g., index.html) with a simple markup:

```
<!DOCTYPE html>
```

```
<html>
    <head>
        <title>Pivot table for Desktop</title>
    </head>
    <body>
    </body>
</html>
```

Step 5. Create the main.js file

In this tutorial, the entry point for the application is main.js. Create a simple main.js file that waits for the application to be ready and then opens the window:

```
const { app, BrowserWindow } = require('electron')

function createWindow () {
    // create the browser window
    let win = new BrowserWindow({
        width: 800,
        height: 600,
        webPreferences: {
            nodeIntegration: true
        }
    })

    // load the previously created index.html file
    win.loadFile('index.html')
}

// open the window when the application is ready
app.whenReady().then(createWindow)
```

Step 6. Install Flexmonster

Install Flexmonster by running this CLI command from the folder containing package.json:

```
flexmonster add flexmonster
```

The add command will install the flexmonster npm package to node_modules/ and add it as an npm dependency to package.json.

Step 7. Add a container for Flexmonster

In the index.html file, add a container for Flexmonster:

```
<body>
    <div id="pivotContainer"></div>
```

```
</body>
```

Step 8. Create a file for a Flexmonster instance

Create a new .js file (e.g., pivot.js) for Flexmonster and include the Flexmonster npm module into it:

```
require('flexmonster');
```

Step 9. Add a Flexmonster instance

Add a new Flexmonster instance to the pivot.js file:

```
const pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: "https://cdn.flexmonster.com/reports/report.json"
});
```

Step 10. Add Flexmonster to the index.html file

Include the script embedding Flexmonster to the index.html file:

```
<div id="pivotContainer"></div>
<script src="./pivot.js"></script>
```

Step 11. Include Flexmonster styles

Include Flexmonster styles into the index.html file. It should be done as follows:

```
<head>
    <title>Pivot table for Desktop</title>
    <link rel="stylesheet" type="text/css"
     href="https://cdn.flexmonster.com/flexmonster.min.css"
    />
</head>
```

Step 12. Run the project

Run the project from the console with the following command:

```
npm start
```

A desktop application with the pivot table embedded will be launched.

The application can be shut down manually with Ctrl+C.

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (/doc/customizing-toolbar/)
- How to add localization (/doc/localizing-component/)
- How to customize the appearance with CSS (/doc/customizing-appearance/)
- How to configure the way that data is exported (/doc/export-and-print/)

3.5.9. Integration with Webpack

This tutorial will help you integrate the pivot table with webpack (https://webpack.js.org/). Follow these steps to set up a simple webpack project using Flexmonster Pivot Table & Charts:

Prerequisites

To run a simple application you will need Node.js and npm. Get it here (https://docs.npmjs.com/getting-started/installing-node) if it’s not already installed on your machine.

Open a terminal/console window and verify that you are running at least node v4.x.x and npm 3.x.x by running node -v and npm -v.

Step 1: Create a new project based on the sample from GitHub

Download the .zip archive with the sample or clone it from GitHub (https://github.com/flexmonster/pivot-webpack) with the following command:

```
git clone https://github.com/flexmonster/pivot-webpack my-proj
cd my-proj
```

Step 2: Install npm packages

Install the npm packages described in package.json:

```
npm install
```

Step 3: Run the sample project

Run in the console:

```
npm start
```

After that, the working sample will be available at http://localhost:8080/.

Project structure

Let’s take a closer look at the folder structure of this application:

- dist/
  - index.html – here we add the <div> container for Flexmonster.
- src/
  - index.js – here we embed our pivot table.
- package.json – contains the description of the npm packages.
- webpack.config.js – the webpack configuration file with standard CSS and font loading.

Take a look at the content of the key files:

index.js

```
// ES2015
import 'flexmonster/flexmonster.min.css';
import Flexmonster from 'flexmonster';

// CommonJS
// require('flexmonster/flexmonster.min.css');
// const Flexmonster = require('flexmonster');

new Flexmonster({
    container: "#pivot",
    toolbar: true
})
```

webpack.config.js

```
const path = require('path');

module.exports = {
    mode: 'development',
    entry: {
        app: './src/index.js',
    },
    devServer: {
        contentBase: './dist'
    },
    output: {
        filename: 'bundle.js',
        path: path.resolve(__dirname, 'dist')
    },
    module: {
        rules: [
            {
                test: /\.css$/,
                use: [
                    'style-loader',
                    'css-loader'
                ]
            },
            {
                test: /\.(woff|woff2|eot|ttf|otf|svg)$/,
                use: [
                    'file-loader'
                ]
            }
        ]
    }
};
```

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Flexmonster / Webpack</title>
</head>
<body>
  <div id="pivot"></div>
  <script src="bundle.js"></script>
</body>
</html>
```

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/)
- How to add localization (https://www.flexmonster.com/doc/localizing-component/)
- How to customize appearance with CSS (https://www.flexmonster.com/doc/customizing-appearance/)
- How to define a format for date and time (https://www.flexmonster.com/doc/date-and-time-formatting/)
- How to set specific options common for all reports (https://www.flexmonster.com/doc/global-object/)
- How to configure the way that data is exported (https://www.flexmonster.com/doc/export-and-print/)

3.5.10. Integration with RequireJS

This tutorial will help you integrate the pivot table with the RequireJS framework (https://requirejs.org/). Follow this guide to set up a simple RequireJS project using Flexmonster Pivot Table & Charts.

Prerequisites

To run a simple application, you will need Node.js and npm. Get it here (https://docs.npmjs.com/getting-started/installing-node) if it's not already installed on your machine.

Step 1. Create a new project based on the sample from GitHub

Download the .zip archive with the sample or clone it from GitHub (https://github.com/flexmonster/pivot-requirejs) with the following command:

```
git clone https://github.com/flexmonster/pivot-requirejs
cd pivot-requirejs
```

Step 2. Install npm packages

Install the npm packages described in package.json:

```
npm install
```

Step 3. Run the sample project

Open the index.html file in the browser to see the result.

Project structure

Let's take a closer look at the folder structure of this application:

- scripts/ – the folder containing application JavaScript files:
  - main.js – the main application file; here we embed Flexmonster.
  - require.js – the RequireJS library.
- index.html – here we add the <div> container for Flexmonster.
- package.json – contains the description of the npm packages.

Take a look at the content of the key files:

scripts/main.js

```
requirejs(["node_modules/flexmonster/flexmonster.full.js"],
function() {
    var pivot = new Flexmonster({
        container: "pivotContainer",
        toolbar: true,
        report: "https://cdn.flexmonster.com/reports/report.json"
    })
})
```

index.html

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <title>Flexmonster Sample Project</title>
      <!-- data-main attribute tells require.js to load
           scripts/main.js after require.js loads. -->
      <script data-main="scripts/main"
              src="scripts/require.js">
      </script>
      <link rel="stylesheet" type="text/css"
            href="node_modules/flexmonster/flexmonster.css">
  </head>
  <body>
    <h1>Flexmonster Sample Project</h1>
    <div id="pivotContainer"></div>
  </body>
</html>
```

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/)
- How to add localization (https://www.flexmonster.com/doc/localizing-component/)
- How to customize the appearance with CSS (https://www.flexmonster.com/doc/customizing-appearance/)
- How to configure the way that data is exported (https://www.flexmonster.com/doc/export-and-print/)

4. Connecting to Data Source

4.1. Supported data sources

Flexmonster supports many popular data sources, including CSV, JSON, SQL databases, etc. This is an overview of the guides for different data sources.

JSON

Flexmonster can handle local and remote JSON files, as well as inline JSON data. See how to use this data source in the section about JSON (https://www.flexmonster.com/doc/json-data-source/).

CSV

You can load data to Flexmonster either from a local or remote CSV file. Our CSV section (https://www.flexmonster.com/doc/csv-data-source/) will guide you through the connection process.

Relational databases

Flexmonster supports the most popular SQL databases: MySQL, Microsoft SQL Server, PostgreSQL, and Oracle.

For more information, see this overview article (https://www.flexmonster.com/doc/connect-to-relational-database/).

Flexmonster Data Server

Flexmonster Data Server is a server-side solution for loading datasets from JSON/CSV files and relational databases. Check out our detailed guides (https://www.flexmonster.com/doc/intro-to-flexmonster-data-server/) that cover different specifics of the Data Server usage.

MongoDB

You can connect to this database using Flexmonster MongoDB Connector – our server-side utility. Learn how to use it in our documentation (https://www.flexmonster.com/doc/mongodb-connector/).

Microsoft Analysis Services

Flexmonster supports two ways of connecting to Analysis Services: via XMLA and Flexmonster Accelerator – our tool designed to increase the data loading speed. Learn more about both approaches in the section about Microsoft Analysis Services (https://www.flexmonster.com/doc/connecting-to-microsoft-analysis-services/).

Custom data source

Connecting to any data source is possible with the custom data source API. It is our custom communication protocol for fetching aggregated data from a server to Flexmonster. For details, refer to our guides (https://www.flexmonster.com/doc/introduction-to-custom-data-source-api/).

Elasticsearch

To learn how to use Flexmonster with the Elasticsearch data source, see our guide on Elasticsearch (https://www.flexmonster.com/doc/connecting-to-elasticsearch/).

Mondrian

You can connect to Pentaho Mondrian using either XMLA or Flexmonster Accelerator. Refer to our documentation (https://www.flexmonster.com/doc/connecting-to-pentaho-mondrian/) for details.

Other data sources

If your data source is not mentioned above, visit the article about other data sources (https://www.flexmonster.com/doc/connecting-to-other-databases/). It describes different options for connecting to any data source.

4.2. JSON

4.2.1. Connecting to JSON

This article illustrates how to build a report based on a JSON data source. The most simple configuration to use a JSON data source is to display data that is already on your page. Alternatively, you can load JSON data from your local file system (see Connect > To local JSON in the Toolbar), refer to a remote JSON file, or add data generated by a server-side script.

To load big JSON files (more than 100 MB), we recommend using Flexmonster Data Server – our powerful tool for processing large datasets. Refer to this tutorial for details: Connecting to JSON using Flexmonster Data Server (/doc/pivot-table-connect-to-json/).

The component supports two JSON formats – an array of objects, where each object is an unordered set of name/value pairs, and an array of arrays, where each sub-array contains ordered values. See the array of objects sample below:

```
var jsonData = [
    {
        "Color" : "green",
        "Country" : "Canada",
        "State" : "Ontario",
        "City" : "Toronto",
        "Price" : 174,
        "Quantity" : 22
    },
    {
        "Color" : "red",
        "Country" : "USA",
        "State" : "California",
        "City" : "Los Angeles",
        "Price" : 166,
        "Quantity" : 19
    }
];
```

The following sample demonstrates what a JSON format with an array of arrays looks like:

```
var jsonData = [
    [
        "Category",
        "Color",
        "Price",
        "Quantity"
    ],
    ["Ice-cream", "red", 23.32, 4],
    ["Ice-cream", "yellow", 4.2, 3],
    ["Sweets", "red", 45.3, 2],
    ["Candies", "orange", 22.65, 7]
];
```

To use JSON data as a data source in the pivot table embedded in your project, follow these steps:

1. If Flexmonster is not yet embedded, set up an empty component in your web page:

   ## In pure JavaScript

   Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

2. Copy the following JSON and paste it into your web page:

```
var jsonData = [
    {
  "Color" : "green",
  "Country" : "Canada",
  "State" : "Ontario",
  "City" : "Toronto",
  "Price" : 174,
     "Quantity" : 22
    },
    {
  "Color" : "red",
  "Country" : "USA",
  "State" : "California",
  "City" : "Los Angeles",
  "Price" : 166,
  "Quantity" : 19
    }
];

var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

3. Add the data property to the report object:

```
var jsonData = [
    {
  "Color" : "green",
  "Country" : "Canada",
  "State" : "Ontario",
  "City" : "Toronto",
  "Price" : 174,
     "Quantity" : 22
    },
    {
  "Color" : "red",
  "Country" : "USA",
  "State" : "California",
  "City" : "Los Angeles",
  "Price" : 166,
  "Quantity" : 19
    }
];

var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
         data: jsonData
     }
    }
});
```

4. Define a slice – fields that go to rows, go to columns and go to measures:

```
var jsonData = [
    {
  "Color" : "green",
  "Country" : "Canada",
  "State" : "Ontario",
  "City" : "Toronto",
  "Price" : 174,
     "Quantity" : 22
    },
    {
  "Color" : "red",
  "Country" : "USA",
  "State" : "California",
  "City" : "Los Angeles",
  "Price" : 166,
  "Quantity" : 19
    }
];

var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
         data: jsonData
     },
     slice: {
         rows: [
             { uniqueName: "Color" },
          { uniqueName: "[Measures]" }
         ],
         columns: [
             { uniqueName: "Country" }
         ],
         measures: [
             {
                    uniqueName: "Price",
              aggregation: "sum"
             }
         ]
     }
    }
});
```

5. Now launch the page from a browser — there you go! A pivot table based on JSON data is embedded into your project. Check out this example on JSFiddle (https://jsfiddle.net/flexmonster/pz431qp5/).

Now you know how to build a report based on our sample JSON. This example (https://jsfiddle.net/flexmonster/scro38mr/) shows how to use a JSON array of arrays. The next step for you is to visualize your JSON data.

How to display non-English characters correctly

If you use an alphabet with special characters you must use UTF-8 encoding. There are two ways to do this:

1. Specify the encoding for your HTML page to UTF-8 using the content-type HTTP header or the corresponding meta tag:

   ```
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   ```

   Content from a database and static JSON files also must be encoded as UTF-8.

2. If you are not able to change the content of your HTML file, you can embed Flexmonster Pivot in a separate JS file with specified UTF-8 encoding.

   ```
   <script src="yourfile.js" charset="UTF-8"></script>
   ```

What’s next?

You may be interested in the following articles:

- Connecting to JSON using Flexmonster Data Server (/doc/pivot-table-connect-to-json/)
- How to define data types in JSON (/doc/data-types-in-json/)
- How to configure which data subset is shown (/doc/slice/)
- How to specify functionality available to customers (/doc/options/)
- How to define conditional formatting (/doc/conditional-formatting/)

4.2.2. Connecting to JSON using Flexmonster Data Server

In the previous article (/doc/json-data-source/), we described how to connect the pivot table to the JSON data source. To gain even better performance while working with JSON datasets, use Flexmonster Data Server (/doc/intro-to-flexmonster-data-server/) – a special server developed by Flexmonster. This server communicates with the client using the custom data source API (/doc/introduction-to-custom-data-source-api/) – our custom communication protocol allowing you to retrieve already aggregated data from a server to Flexmonster Pivot.

Flexmonster Data Server has the following advantages:

- **Reduced browser’s memory usage.** Since loading, processing, filtering, and aggregating of the data is made on the server side, Flexmonster Pivot gets the data in a ready-to-show format. Thus, the data is visualized faster, and the browser’s resources are used more efficiently.
- **Larger datasets**. Despite Flexmonster Pivot can connect to a JSON file directly, the browser’s capability limits the size of a file (on average, up to 100 MB). Flexmonster Data Server can process files more than 1GB — the maximum file size depends on your RAM capacity and the number of unique members in the dataset.
- **Built-in security configurations.** The Data Server allows establishing authorized access to data and managing security in a convenient way.
- **Cross-platform solution**. The Data Server is available for popular operating systems: Windows, Ubuntu/Linux, and macOS.

Flexmonster Data Server supports only a specific JSON format – an array of objects, where each object is an unordered set of "key": "value" pairs. Here is an example:

```
[
   {
       "Color" : "green",
       "Country" : "Canada",
       "State" : "Ontario",
       "City" : "Toronto",
       "Price" : 174,
       "Quantity" : 22
   },
   ...
]
```

Prerequisites

To download the Data Server, you will need Flexmonster CLI (https://www.flexmonster.com/doc/cli-overview/) — a command-line interface tool for Flexmonster. If needed, install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

After that, a new flexmonster command will be available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

Now follow the steps below to connect to JSON using the Data Server.

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Download Flexmonster Data Server

The previous step demonstrated how to configure Flexmonster Pivot. Now it’s time to set up Flexmonster Data Server.

Get the Data Server with the following CLI command:

```
flexmonster add fds
```

The flexmonster add fds command will download the .zip archive with Flexmonster Data Server and automatically unpack the files in the current folder. As a result, the flexmonster-data-server/ folder will appear in your working directory.

The Data Server can be configured in the flexmonster-data-server/flexmonster-config.json file. To learn more about all the available configurations, see the configurations reference (/doc/configurations-reference/).

To set configurations needed for the JSON data source, follow the steps below.

Step 3. Configure the data source type

Set the "Type" property in flexmonster-config.json to "json". It should be done as follows:

```
"DataSources": [
    {
        "Type": "json"
    }
],
```

Step 4. Define the dataset

Create an index for the file with your data. "Path" is the path to the file. For example:

```
"DataSources": [
    {
        "Type": "json",
        "Indexes": {
            "index_json": {
                "Path": "./data/data.json"
            }
        }
    }
],
```

In a similar way, additional indexes can be specified:

```
"Indexes": {
    "index_json": {
        "Path": "./data/data.json"
    },
    "another_index_json": {
        "Path": "./data/another_data.json"
    }
}
```

"index_json" and "another_index_json" are dataset identifiers. They will be used to configure the data source on the client side.

Step 5. Run the server

Start Flexmonster Data Server by running the following command in the console:

on Windows

```
flexmonster-data-server.exe
```

on macOS and Ubuntu/Linux

```
./flexmonster-data-server
```

As soon as you start Flexmonster Data Server, it automatically preloads the data specified in the "Indexes" property. Thus, when Flexmonster Pivot requests the data, the server responds with already preloaded data.

The Data Server keeps preloaded data in the server’s RAM, so the number of indexes you can specify is limited by the server’s RAM amount.

Step 6. Configure the report

On the client side, the report should be configured as follows:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:9500",
            index: "index-json"
        }
    }
});
```

index must match the name of the index defined in step 4 (e.g., "index_json").

When Flexmonster Pivot requests the data, Flexmonster Data Server server sends the response and then caches it. In case the component sends the same request once again, the server responds with the data from its cache. The Data Server clears the cache when restarted.

The Data Server’s cache has a memory limit. When the cache does not have enough memory for a new response, the Data Server server deletes one of the previously cached responses.

You can manage the cache size via the “CacheSizeLimit” property (https://www.flexmonster.com/doc/configurations-reference/#CacheSizeLimit).

What’s next?

You may be interested in the following articles:

- Flexmonster Data Server configurations reference (/doc/configurations-reference/)
- Flexmonster Data Server security guide (/doc/security-and-authorization-guide/)
- How to define data types in JSON (/doc/mapping/)
- How to configure which data subset is shown (https://www.flexmonster.com/doc/slice/)
- How to specify functionality available to customers (https://www.flexmonster.com/doc/options/)

4.2.3. Data types in JSON

We recommend using the Mapping Object (https://www.flexmonster.com/doc/mapping/) to customize the representation and structure of JSON data.

The Mapping Object supports:

- Setting data types in JSON.
- Specifying a list of aggregations available for a field.
- Hiding UI filters for a specific field.
- Formatting a certain date field.
- And many other configs.

Additionally, the mapping allows separating your data from its representation.

For details on setting the mapping, see our guide (https://www.flexmonster.com/doc/mapping/).

Alternatively, the first object of the input JSON array can be used for these needs.

Here is the list of its supported properties:

- type – String. The data type. Can be:
  - "string" – the field contains string data. The field’s members will be sorted as strings.
  - "number" – the field contains numeric data. You will be able to aggregate it with all available aggregations. It will be sorted as numeric data.
  - "month" – the field contains months. Note that if the field stores month names only (in either short or full form), the field will be recognized by Flexmonster as a field of the "month" type automatically. If the field contains custom month names, specify its type as "month" explicitly.
  - "weekday" – the field contains days of the week.
  - "date" – the field is a date. Such fields will be split into 3 different fields: Year, Month, Day.
  - "year/month/day" – the field is a date. You will see such dates as hierarchies: Year > Month > Day.
  - "year/quarter/month/day" – the field is a date. You will see such dates as hierarchies: Year > Quarter > Month > Day.
  - "date string" – the field is a date. Such fields will be formatted using a date pattern (default is "dd/MM/yyyy").
  - "datetime" – the field is a date (numeric data). Such fields will be formatted using "dd/MM/yyyy HH:mm:ss" pattern. Min, max, count, and distinctcount aggregations can be applied to it.
  - "time" – the field is a time (numeric data). Such fields will be formatted using "HH:mm:ss" pattern.

    - "id" – the field is an id of the record. Such fields are used for editing data. This field will not be shown in the Field List.
    - "property" – the field for setting member properties. This field will not be shown in the Field List. For example, it can be used to associate a productID with a product. See the example (https://jsfiddle.net/flexmonster/nm09d7zh/).
- hierarchy – String. The hierarchy's name. When configuring hierarchies, specify this property to mark the field as a level of a hierarchy or as a member property of a hierarchy (in this case, the type parameter should be set to "property").
- parent – String. The unique name of the parent level. This property is necessary to specify if the field is a level of a hierarchy and has a parent level.
- isMeasure (optional) – Boolean. When set to true, the field can be selected only as a measure. The isMeasure property should be used only with the strictDataTypes option (https://www.flexmonster.com/api/options-object/#strictDataTypes). *Default value: false*.

For example, you can add the following first object in a JSON array and see how it changes the report:

```
var jsonData = [
    {
        "Color": {type: "string"},
        "Country": {
            type: "string",
            hierarchy: "Geography"
        },
        "State": {
            type: "string",
            hierarchy: "Geography",
            parent: "Country"
        },
        "City": {
            type: "string",
            hierarchy: "Geography",
            parent: "State"
        },
        "Price": {type: "number"},
        "Quantity": {type: "number"}
    },
    {
        "Color" : "green",
        "Country" : "Canada",
        "State" : "Ontario",
        "City" : "Toronto",
        "Price" : 174,
        "Quantity" : 22
    },
    {
        "Color" : "red",
        "Country" : "USA",
        "State" : "California",
        "City" : "Los Angeles",
        "Price" : 166,
        "Quantity" : 19
    }
];

var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            data: jsonData
        },
        slice: {
            rows: [
                { uniqueName: "Color" },
                { uniqueName: "[Measures]" }
            ],
            columns: [
                { uniqueName: "Geography" }
            ],
            measures: [
                { uniqueName: "Price", aggregation: "sum" }
            ]
        }
    }
});
```

Try it on JSFiddle (https://jsfiddle.net/flexmonster/mr0Lwkab/).

Note: if you use a JSON array of arrays you can also add the first object. In this case, you do not need to specify hierarchies in the first sub-array. Check out a live example (https://jsfiddle.net/flexmonster/Lgdjfzxd/).

Automatic type selection in JSON

Flexmonster selects field types automatically. If needed, you can define only necessary types of fields in both mapping and the first JSON object.

In the example below, only the type of "Date" is set explicitly. Types of "Country" and "Price" will be set automatically as "string" and "number", respectively:

```
var jsonData = [
    {
        "Date": { type: "date string" }
    },
    {
        "Date" : "2021-05-25",
        "Country" : "Canada",
        "Price" : 174
    },
    {
        "Date" : "2021-03-18",
        "Country" : "USA",
        "Price" : 166
    }
];

var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            data: jsonData
        }
    }
});
```

Supported date formats

To make date fields be interpreted as a date, you must define the data type as a date. For example, "type": "date", "type": "date string", "type": "datetime", "type": "year/month/day" or "type": "year/quarter/month/day". Additionally, data from these fields should have a special date format to be understood properly.

Flexmonster supports ISO 8601 (https://www.w3.org/TR/NOTE-datetime) as an input date format:

- "2021-05-25" – Date.
- "2021-05-25T21:30:00" – Date and time.
- "2021-05-25T21:30:00+03:00" – Date and time with a time zone.

Other formats aren't officially supported and may have unexpected results.

4.3. CSV

4.3.1. Connecting to CSV

This article illustrates how to build a report based on a CSV data source. A CSV data source is commonly used to display data exported from other sources (for example from Excel). Alternatively, you can load CSV data from a file (see Connect > To local CSV in the Toolbar), refer to a remote CSV file, or add data generated by a server-side script.

To load big CSV files (more than 100 MB), we recommend using Flexmonster Data Server. This server-side tool was designed to handle large datasets. Learn more here: Connecting to CSV using Flexmonster Data Server (/doc/pivot-table-connect-to-csv/).

Complete the following steps to use CSV data as a data source in the pivot table embedded in your project.

1. If Flexmonster is not yet embedded, set up an empty component in your web page:

   ### In pure JavaScript

   Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

   ```
   var pivot = new Flexmonster({
       container: "pivotContainer",
       toolbar: true
   ```

```
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

2. For this example, we will use a CSV file called data.csv. For simplicity, copy this file to the flexmonster/ folder.

```
Category,Size,Color,Destination,Business Type,Country,Price,Quantity,Discount
Accessories,262 oz,red,Australia,Specialty Bike Shop,Australia,174,225,23
Accessories,214 oz,yellow,Canada,Specialty Bike Shop,Canada,502,90,17
Accessories,147 oz,white,France,Specialty Bike Shop,France,242,855,37
Accessories,112 oz,yellow,Germany,Specialty Bike Shop,Germany,102,897,42
```

3. Add the filename property to the report object:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            filename: "data.csv"
        }
    }
});
```

If CSV fields in your file are separated by anything other than , or ;, specify the fieldSeparator property (https://www.flexmonster.com/doc/data-source/#fieldSeparator) in the report.

4. Define a slice – fields that go to rows, go to columns and go to measures:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            filename: "data.csv"
        },
        slice: {
            rows: [
                { uniqueName: "Color" },
                { uniqueName: "[Measures]" }
            ],
            columns: [
                { uniqueName: "Country" }
            ],
            measures: [
                { uniqueName: "Price", aggregation: "sum" }
            ]
        }
    }
});
```

5. Now launch the page from a browser — there you go! A pivot table based on CSV data is embedded into your project. Open this example on JSFiddle (https://jsfiddle.net/flexmonster/5jfc7gjy/).

Now you know how to build a report based on our sample CSV. The next step for you is to visualize your CSV data.

How to display non-English characters correctly

If you use an alphabet with special characters you must use UTF-8 encoding. There are two ways to do this:

1. Specify the encoding for your HTML page to UTF-8 using the content-type HTTP header or the

corresponding meta tag:

```
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
```

Content from a database and static CSV files also must be encoded as UTF-8.

2. If you are not able to change the content of your HTML file, you can embed Flexmonster Pivot in a separate JS file with specified UTF-8 encoding.

```
<script src="yourfile.js" charset="UTF-8"></script>
```

What’s next?

You may be interested in the following articles:

- Connecting to CSV using Flexmonster Data Server (/doc/pivot-table-connect-to-csv/)
- How to define data types in CSV (/doc/data-types-in-csv/)
- How to configure which data subset is shown (/doc/slice/)
- How to specify functionality available to customers (/doc/options/)
- How to define conditional formatting (/doc/conditional-formatting/)

4.3.2. Connecting to CSV using Flexmonster Data Server

In the previous article (/doc/csv-data-source/), we described how to connect the pivot table to the CSV data source. To gain even better performance while working with CSV datasets, use Flexmonster Data Server (/doc/intro-to-flexmonster-data-server/) – a special server developed by Flexmonster. This server communicates with the client using the custom data source API (/doc/introduction-to-custom-data-source-api/) – our custom communication protocol allowing you to retrieve already aggregated data from a server to Flexmonster Pivot.

Flexmonster Data Server has the following advantages:

- **Reduced browser’s memory usage.** Since loading, processing, filtering, and aggregating of the data is made on the server side, Flexmonster Pivot gets the data in a ready-to-show format. Thus, the data is visualized faster, and the browser’s resources are used more efficiently.
- **Larger datasets.** Despite Flexmonster Pivot can connect to a CSV file directly, the browser’s capability limits the size of a file (on average, up to 100 MB). Flexmonster Data Server can process files more than 1GB — the maximum file size depends on your RAM capacity and the number of unique members in the dataset.
- **Built-in security configurations.** The Data Server allows establishing authorized access to data and managing security in a convenient way.
- **Cross-platform solution.** The Data Server is available for popular operating systems: Windows, Ubuntu/Linux, and macOS.

Prerequisites

To download the Data Server, you will need Flexmonster CLI (https://www.flexmonster.com/doc/cli-overview/) — a command-line interface tool for Flexmonster. If needed, install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

After that, a new flexmonster command will be available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

Now follow the steps below to connect to CSV using the Data Server.

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Download Flexmonster Data Server

The previous step demonstrated how to configure Flexmonster Pivot. Now it’s time to set up Flexmonster Data Server.

Get the Data Server with the following CLI command:

```
flexmonster add fds
```

The flexmonster add fds command will download the .zip archive with Flexmonster Data Server and automatically unpack the files in the current folder. As a result, the flexmonster-data-server/ folder will appear in your working directory.

The Data Server can be configured in the flexmonster-data-server/flexmonster-config.json file. To learn more about all the available configurations, see the configurations reference (/doc/configurations-reference/).

To set configurations needed for the CSV data source, follow the steps below.

Step 3. Configure the data source type

Set the "Type" property in flexmonster-config.json to "csv". It should be done as follows:

```
"DataSources": [
    {
        "Type": "csv"
    }
],
```

Step 4. Define the dataset

Create an index for the file with your data. "Path" is the path to the file. For example:

```
"DataSources": [
    {
        "Type": "csv",
        "Indexes": {
            "index_csv": {
                "Path": "./data/data.csv"
            }
        }
    }
],
```

In a similar way, additional indexes can be specified:

```
"Indexes": {
    "index_csv": {
        "Path": "./data/data.csv"
    },
    "another_index_csv": {
        "Path": "./data/another_data.csv"
    }
}
```

"index_csv" and "another_index_csv" are dataset identifiers. They will be used to configure the data source on the client side.

Step 5. (optional) Specify the field delimiter

If CSV fields are separated by anything other than ",", the "Delimiter" parameter should be specified:

```
"Indexes": {
    "index_csv": {
        "Path": "./data/data.csv",
        "Delimiter": ";"
    }
}
```

Step 6. (optional) Specify the decimal separator

If decimal parts of numbers in the data are separated by anything other than ".", the "DecimalSeparator" parameter should be specified:

```
"Indexes": {
    "index_csv": {
        "Path": "./data/data.csv",
        "Delimiter": ";",
        "DecimalSeparator": ","
   }
}
```

Step 7. (optional) Specify the thousand separator

If the data contains numbers where groups of digits are separated by anything other than ",", the "ThousandSeparator" parameter should be specified:

```
"Indexes": {
  "index_csv": {
    "Path": "./data/data.csv",
    "Delimiter": ";",
    "DecimalSeparator": ",",
    "ThousandSeparator": "."
  }
}
```

Step 8. Run the server

Start Flexmonster Data Server by running the following command in the console:

on Windows

```
flexmonster-data-server.exe
```

on macOS and Ubuntu/Linux

```
./flexmonster-data-server
```

As soon as you start Flexmonster Data Server, it automatically preloads the data specified in the "Indexes" property. Thus, when Flexmonster Pivot requests the data, the server responds with already preloaded data.

The Data Server keeps preloaded data in the server’s RAM, so the number of indexes you can specify is limited by the server’s RAM amount.

Step 9. Configure the report

On the client side, the report should be configured as follows:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:9500",
            index: "index-csv"
        }
    }
});
```

index must match the name of the index defined in step 4 (e.g., "index_csv").

When Flexmonster Pivot requests the data, Flexmonster Data Server server sends the response and then caches it. In case the component sends the same request once again, the server responds with the data from its cache. The Data Server clears the cache when restarted.

The Data Server's cache has a memory limit. When the cache does not have enough memory for a new response, the Data Server server deletes one of the previously cached responses.

You can manage the cache size via the "CacheSizeLimit" property (https://www.flexmonster.com/doc/configurations-reference/#CacheSizeLimit).

What's next?

You may be interested in the following articles:

- Flexmonster Data Server configurations reference (/doc/configurations-reference/)
- Flexmonster Data Server security guide (/doc/security-and-authorization-guide/)
- How to define data types in CSV (/doc/mapping/)
- How to configure which data subset is shown (/doc/slice/)
- How to specify functionality available to customers (/doc/options/)

4.3.3. Data types in CSV

We recommend using the Mapping Object (https://www.flexmonster.com/doc/mapping/) to customize the representation and structure of CSV data. See the migration guide from CSV prefixes to the mapping (#migrate-to-mapping).

Alternatively, you can use special prefixes for column names to indicate how data should be interpreted by Flexmonster Pivot. Note that the Mapping Object provides more options than the approach with prefixes.

Migrating from CSV prefixes to the Mapping Object

The Mapping Object (/api/mapping-object/) is a simple and convenient way of defining the fields' data type, and we recommend using it instead of CSV prefixes.

For easy migration from CSV prefixes to the mapping, see the migration table below:

CSV prefix	Mapping type	Description
+	"string"	The field is a dimension.
-	"number"	The field is a value.
m+	"month"	The field stores months.
w+	"weekday"	The field stores days of the week.

d+	"date"	The field stores a date. The field of this type is split into 3 different fields: Year, Month, and Day.
D+	"year/month/day"	The field stores a date. It’s displayed as a multilevel hierarchy with the following levels: Year > Month > Day.
D4+	"year/quarter/month/day"	The field is a date. It’s displayed as a multilevel hierarchy with the following levels: Year > Quarter > Month > Day.
ds+	"date string"	The field stores a date. It can be formatted using the datePattern option (default is "dd/MM/yyyy").
t+	"time"	The field stores time.
dt+	"datetime"	The field stores a date. It can be formatted using the dateTimePattern option (default is "dd/MM/yyyy HH:mm:ss").
id+	"id"	The field is an id. The field of this type can be used for editing data. It’s not shown in the Field List.

Supported CSV prefixes

Here is the list of supported prefixes that can be used to customize the CSV data:

- + – the field is a dimension.
- - – the field is a value.
- m+ – the field is a month. Note that if the field stores month names only (in either short or full form), the field will be recognized by Flexmonster as a field of the "m+" type automatically. If the field contains custom month names, specify its type as "m+" explicitly.
- w+ – the field is a day of the week.
- d+ – the field is a date. Such fields will be split into 3 different fields: Year, Month, and Day. Date formats that are supported by Flexmonster Pivot are described below.
- D+ – the field is a date. You will see these dates as a hierarchy: Year > Month > Day.
- D4+ – the field is a date. You will see these dates as a hierarchy: Year > Quarter > Month > Day.
- ds+ – the field is a date. Such fields will be formatted using a date pattern (default is "dd/MM/yyyy").
- t+ – the field is a time (measure). Such fields will be formatted using "HH:mm:ss" pattern.
- dt+ – the field is a date (measure). Such fields will be formatted using "dd/MM/yyyy HH:mm:ss" pattern.
- id+ – the field is an id of the record.

Here is the minimal CSV data that will treat Year as a dimension, rather than a numeric measure:

```
Country, +Year, Sales
US, 2010, 200
UK, 2010, 100
```

Supported date formats

To make date column be interpreted as a date, use prefixes d+, D+, and D4+ for CSV columns. Additionally, data from these columns should have a special date format to be understood properly.

Flexmonster supports ISO 8601 (https://www.w3.org/TR/NOTE-datetime) as an input date format:

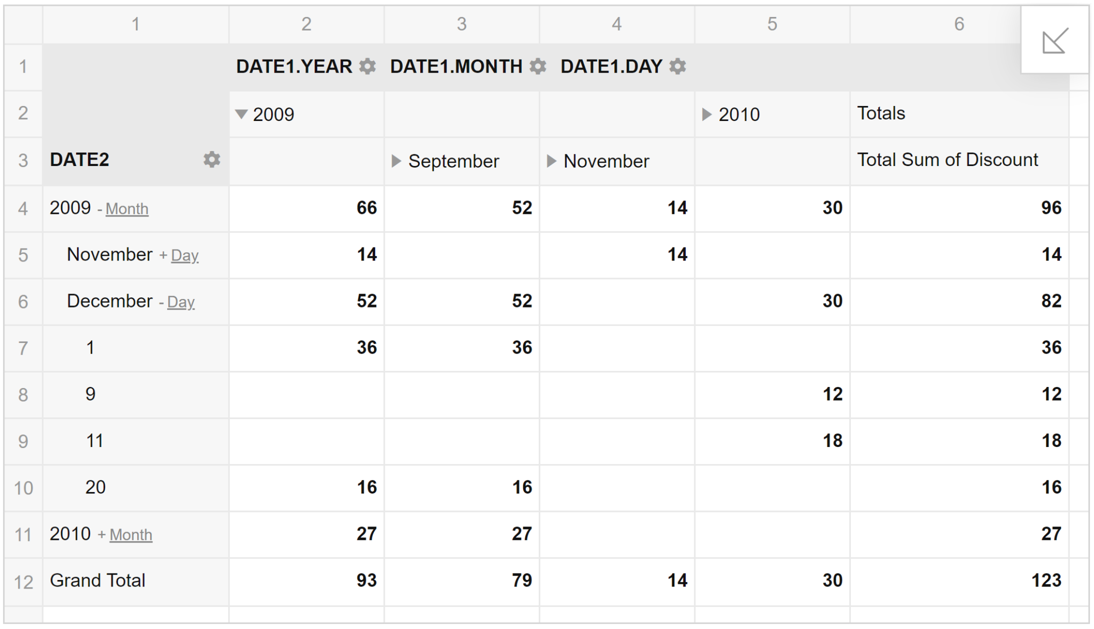
- "2021-05-25" – Date.
- "2021-05-25T21:30:00" – Date and time.
- "2021-05-25T21:30:00+03:00" – Date and time with a time zone.

Other formats aren’t officially supported and may have unexpected results.

Here is an example of CSV data with date columns – Date1 and Date2:

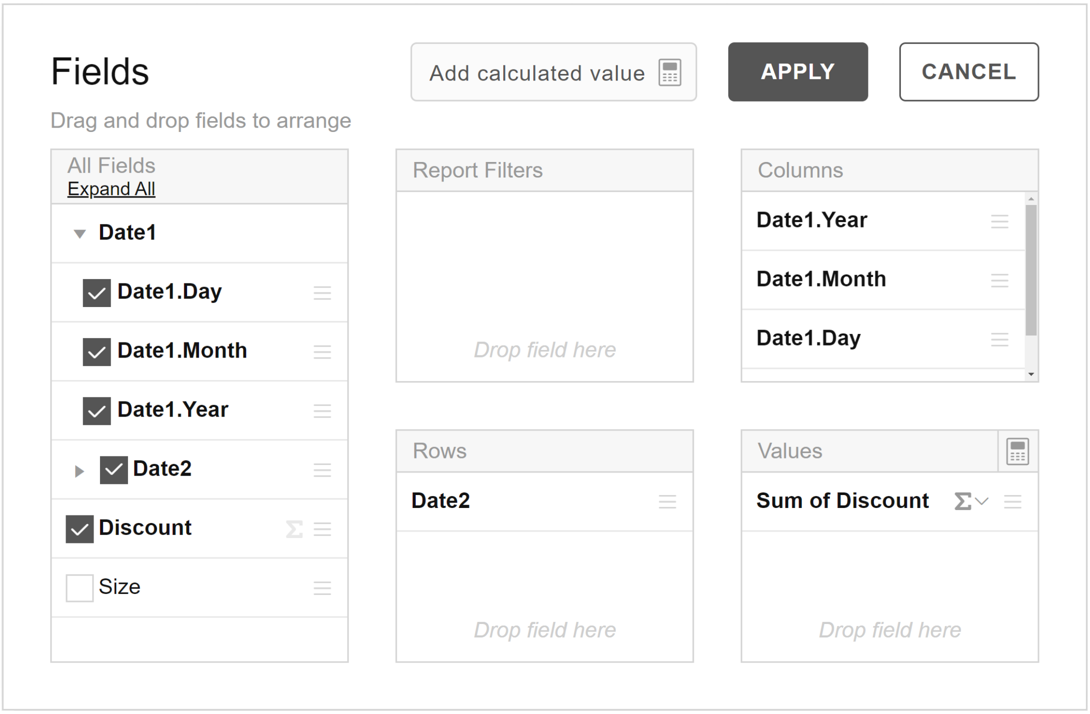
```
Size, Discount, d+Date1, D+Date2
214 oz, 14, 2009-11-01, 2009-11-09
214 oz, 12, 2010-12-09, 2009-12-09
212 oz, 36, 2009-09-01, 2009-12-01
212 oz, 27, 2009-09-01, 2010-12-02
212 oz, 18, 2010-11-09, 2009-12-11
212 oz, 16, 2009-09-01, 2009-12-20
```

The pivot table based on this CSV will look as follows:



	1	2	3	4	5	6
1		DATE1.YEAR	DATE1.MONTH	DATE1.DAY		
2		▼ 2009			▶ 2010	Totals
3	DATE2		▶ September	▶ November		Total Sum of Discount
4	2009 - Month	66	52	14	30	96
5	November + Day	14		14		14
6	December - Day	52	52		30	82
7	1	36	36			36
8	9				12	12
9	11				18	18
10	20	16	16			16
11	2010 + Month	27	27			27
12	Grand Total	93	79	14	30	123

As you can see, the Date1 column with prefix d+ is split into three separate fields — Year, Month, and Day. In the Field List, the Date1 column will look as follows:



The Date2 column with D+ prefix is interpreted as a hierarchy that can be drilled down to months and days.

4.4. Database

4.4.1. Connecting to SQL databases using Flexmonster Data Server

This section illustrates how to connect to a relational database using Flexmonster Data Server (/doc/intro-to-flexmonster-data-server/) – a special server developed by Flexmonster. This server communicates with the client using the custom data source API (/doc/introduction-to-custom-data-source-api/) – our custom communication protocol allowing you to retrieve already aggregated data from a server to Flexmonster Pivot.

This approach has the following advantages:

- **You have full control over data loading.** Flexmonster Data Server allows splitting the data from your database into different subsets, so you can work with that part of the data you need at the time. In addition, this reduces the amount of data filtrations performed by the server so it will work faster.
- **Saving browser's resources.** Loading, processing, filtering, and aggregating of the data is made on the server side, so Flexmonster gets the ready-to-show data. Thus, the data is visualized faster, and the browser's resources are used more efficiently.
- **A convenient way of connecting to a database.** Though there are several ways of fetching data from a database, the speed and capacity of the Data Server make it the best way to load the data from a database.
- **Built-in security configurations.** The Data Server allows establishing authorized access to data and managing security in a convenient way.
- **Cross-platform solution**. Flexmonster Data Server is available for popular operating systems: Windows, Ubuntu/Linux, and macOS.

Supported databases

- MySQL
- Microsoft SQL Server
- PostgreSQL
- Oracle
- Microsoft Azure SQL

To connect to MongoDB, use Flexmonster MongoDB Connector (/doc/mongodb-connector/). If you use other NoSQL databases or data warehouses, refer to our Connecting to other data sources (https://www.flexmonster.com/doc/connecting-to-other-data-sources//) tutorial.

Guides

With Flexmonster Data Server, you can connect to a MySQL, Microsoft SQL Server, PostgreSQL, Oracle, or Microsoft Azure SQL database. Follow detailed tutorials for each database:

- Connecting to a MySQL database (/doc/connect-to-mysql-database/)
- Connecting to a Microsoft SQL Server database (/doc/connect-to-mssql-database/) (works for Microsoft Azure SQL as well)
- Connecting to a PostgreSQL database (/doc/connect-to-postgresql-database/)
- Connecting to an Oracle database (/doc/connect-to-oracle-database/)

4.4.2. Connecting to a MySQL database using Flexmonster Data Server

This tutorial describes how to connect to a MySQL database using Flexmonster Data Server (/doc/intro-to-flexmonster-data-server/) – a special server developed by Flexmonster. This server communicates with the client using the custom data source API (/doc/introduction-to-custom-data-source-api/) – our custom communication protocol allowing you to retrieve already aggregated data from a server to Flexmonster Pivot.

Prerequisites

To download the Data Server, you will need Flexmonster CLI (https://www.flexmonster.com/doc/cli-overview/) — a command-line interface tool for Flexmonster. If needed, install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

After that, a new flexmonster command will be available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

Now follow the steps below to connect to MySQL using the Data Server.

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Download Flexmonster Data Server

The previous step demonstrated how to configure Flexmonster Pivot. Now it’s time to set up Flexmonster Data Server.

Get the Data Server with the following CLI command:

```
flexmonster add fds
```

The flexmonster add fds command will download the .zip archive with Flexmonster Data Server and automatically unpack the files in the current folder. As a result, the flexmonster-data-server/ folder will appear in your working directory.

The Data Server can be configured in the flexmonster-data-server/flexmonster-config.json file. To learn more about all the available configurations, see the configurations reference (/doc/configurations-reference/).

To set configurations needed for a MySQL database, follow the steps below.

Step 3. Configure the data source type

Set the "Type" property in flexmonster-config.json to "database". It should be done as follows:

```
"DataSources": [
    {
        "Type": "database"
    }
],
```

Step 4. Configure the database type

Specify the name of the database you want to use by adding the "DatabaseType" configuration to the flexmonster-config.json file. In this case, the configuration value should be "mysql":

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "mysql"
    }
]
```

Step 5. Configure the connection with the database

To enable the server to fetch data from your database, you have to provide the connection string to the database. The connection string should be added to flexmonster-config.json as follows:

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "mysql",
        "ConnectionString":
 "Server=localhost;Port=3306;Uid=root;Pwd=password;Database=database_name"
    }
]
```

Have a look at the examples of different connection strings for MySQL (https://www.connectionstrings.com/mysql-connector-net-mysqlconnection/).

Step 6. Define the data subset

In the flexmonster-config.json file, create an index for the subset of data you want Flexmonster Pivot to visualize. "Query" is an SQL query for the data. For example:

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "mysql",
        "ConnectionString":
 "Server=localhost;Port=3306;Uid=root;Pwd=password;Database=database_name",
        "Indexes": {
            "index_database": {
                "Query": "SELECT * FROM tablename"
            }
        }
    }
]
```

"index_database" is a dataset identifier. It will be used to configure the data source on the client side.

In a similar way, additional indexes can be specified:

```
"Indexes": {
    "index_database": {
        "Query": "SELECT * FROM tablename"
    },
    "another_index_database": {
        "Query": "SELECT column FROM tablename"
    }
}
```

Step 7. Run the server

To start the Data Server, run the following command in the console:

on Windows

```
flexmonster-data-server.exe
```

on macOS and Ubuntu/Linux

```
./flexmonster-data-server
```

As soon as you start Flexmonster Data Server, it automatically preloads the data specified in the "Indexes" property. Thus, when Flexmonster Pivot requests the data, the server responds with already preloaded data.

The Data Server keeps preloaded data in the server's RAM, so the number of indexes you can specify is limited by the server's RAM amount.

Step 8. Configure the report

On the client side, the report should be configured as follows:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:9500",
            index: "index-database"
        }
    }
});
```

index must match the name of the index defined in step 6 (e.g., "index_database").

When Flexmonster Pivot requests the data, Flexmonster Data Server server sends the response and then caches it. In case the component sends the same request once again, the server responds with the data from its cache. The Data Server clears the cache when restarted.

The Data Server's cache has a memory limit. When the cache does not have enough memory for a new response, the Data Server server deletes one of the previously cached responses.

You can manage the cache size via the "CacheSizeLimit" property (https://www.flexmonster.com/doc/configurations-reference/#CacheSizeLimit).

What's next?

You may be interested in the following articles:

- Flexmonster Data Server configurations reference (/doc/configurations-reference/)
- Flexmonster Data Server security guide (/doc/security-and-authorization-guide/)
- How to configure which data subset is shown (/doc/slice/)
- How to specify functionality available to customers (/doc/options/)
- How to define number formatting (/doc/number-formatting/)

4.4.3. Connecting to a Microsoft SQL Server database using Flexmonster Data Server

This tutorial describes how to connect to a **Microsoft SQL Server** database using Flexmonster Data Server (/doc/intro-to-flexmonster-data-server/) – a special server developed by Flexmonster. This server communicates with the client using the custom data source API (/doc/introduction-to-custom-data-source-api/) – our custom communication protocol allowing you to retrieve already aggregated data from a server to Flexmonster Pivot.

The approach described below can also be used for connecting to **Microsoft Azure SQL** databases.

Prerequisites

To download the Data Server, you will need Flexmonster CLI (https://www.flexmonster.com/doc/cli-overview/) — a command-line interface tool for Flexmonster. If needed, install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

After that, a new flexmonster command will be available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

Now follow the steps below to connect to Microsoft SQL Server using the Data Server.

To learn specifics of connecting to a remote SQL Server database, see this section (#connect-to-remote-db).

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Download Flexmonster Data Server

The previous step demonstrated how to configure Flexmonster Pivot. Now it’s time to set up Flexmonster Data Server.

Get the Data Server with the following CLI command:

```
flexmonster add fds
```

The flexmonster add fds command will download the .zip archive with Flexmonster Data Server and automatically unpack the files in the current folder. As a result, the flexmonster-data-server/ folder will appear in your working directory.

The Data Server can be configured in the flexmonster-data-server/flexmonster-config.json file. To learn more about all the available configurations, see the configurations reference (/doc/configurations-reference/).

To set configurations needed for a Microsoft SQL Server database, follow the steps below.

Step 3. Configure the data source type

Set the "Type" property in flexmonster-config.json to "database". It should be done as follows:

```
"DataSources": [
    {
        "Type": "database"
    }
],
```

Step 4. Configure the database type

Specify the name of the database you want to use by adding the "DatabaseType" configuration to the flexmonster-config.json file. In this case, the configuration value should be "mssql":

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "mssql"
    }
]
```

Step 5. Configure the connection with the database

To enable the server to fetch data from your database, you have to provide the connection string to the database. The connection string should be added to flexmonster-config.json as follows:

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "mssql",
        "ConnectionString":
 "Server=(localdb)\\MSSQLLocalDB;Uid=root;Pwd=password;Database=database_name"
    }
]
```

Have a look at the examples of different connection strings for Microsoft SQL Server (https://www.connectionstrings.com/sqlconnection/) and Microsoft Azure SQL (https://www.connectionstrings.com/azure-sql-database/).

Step 6. Define the data subset

In the flexmonster-config.json file, create an index for the subset of data you want Flexmonster Pivot to visualize. "Query" is an SQL query for the data. For example:

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "mssql",
        "ConnectionString":
 "Server=(localdb)\\MSSQLLocalDB;Uid=root;Pwd=password;Database=database_name",
        "Indexes": {
            "index_database": {
                "Query": "SELECT * FROM tablename"
            }
        }
    }
]
```

"index_database" is a dataset identifier. It will be used to configure the data source on the client side.

In a similar way, additional indexes can be specified:

```
"Indexes": {
    "index_database": {
        "Query": "SELECT * FROM tablename"
    },
    "another_index_database": {
        "Query": "SELECT column FROM tablename"
    }
}
```

Step 7. Run the server

To start the Data Server, run the following command in the console:

on Windows

```
flexmonster-data-server.exe
```

on macOS and Ubuntu/Linux

```
./flexmonster-data-server
```

As soon as you start Flexmonster Data Server, it automatically preloads the data specified in the "Indexes" property. Thus, when Flexmonster Pivot requests the data, the server responds with already preloaded data.

The Data Server keeps preloaded data in the server’s RAM, so the number of indexes you can specify is limited by the server’s RAM amount.

Step 8. Configure the report

On the client side, the report should be configured as follows:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:9500",
            index: "index-database"
        }
    }
});
```

index must match the name of the index defined in step 6 (e.g., "index_database").

When Flexmonster Pivot requests the data, Flexmonster Data Server server sends the response and then caches it. In case the component sends the same request once again, the server responds with the data from its cache. The Data Server clears the cache when restarted.

The Data Server’s cache has a memory limit. When the cache does not have enough memory for a new response, the Data Server server deletes one of the previously cached responses.

You can manage the cache size via the “CacheSizeLimit” property (https://www.flexmonster.com/doc/configurations-reference/#CacheSizeLimit).

Connecting to a remote Microsoft SQL Server database

To enable a remote connection to a Microsoft SQL Server database, you should properly configure your SQL Server instance.

Follow this guide to set up the server so that it allows remote connections. For more details on the SQL Server configuration, refer to this guide (https://knowledgebase.apexsql.com/configure-remote-access-connect-remote-sql-server-instance-apexsql-tools/).

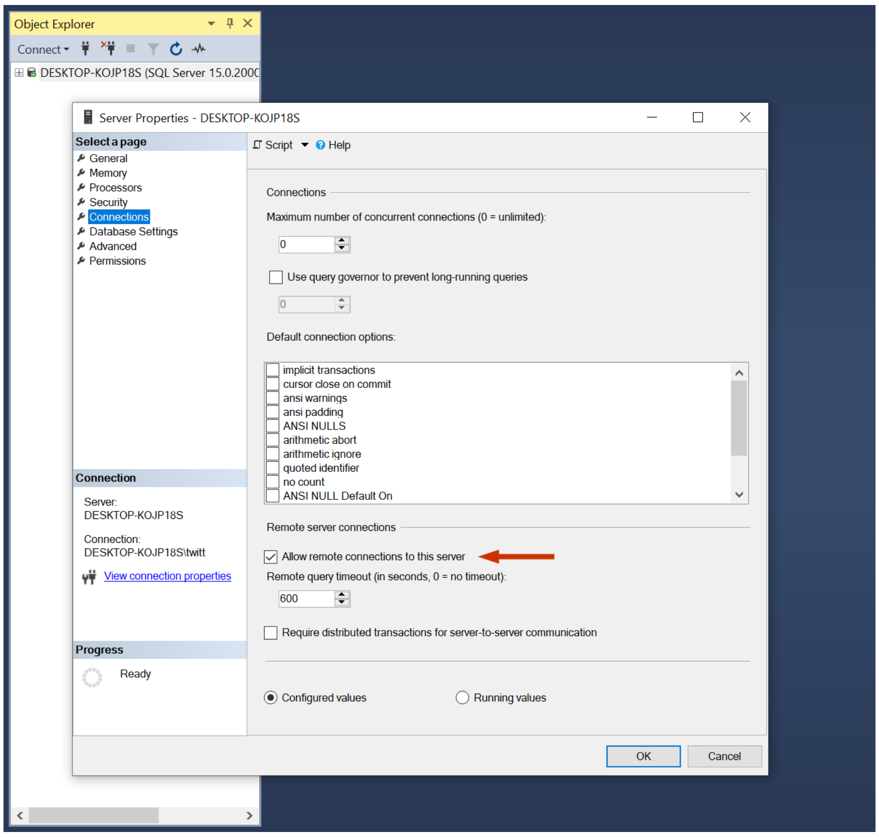
Step 1. Specify the connection string to your database in the flexmonster-config.json file:

```
"DataSources": [
  {
    "Type": "database",
    "DatabaseType": "mssql",
    "ConnectionString": "Server=<server_IP>\\MSSQLSERVER, port;
    Uid=root; Pwd=password; Database=database_name",
    "Indexes": {
      ...
    }
  }
]
```

port can be omitted if your SQL Server instance uses the default 1433 port. Otherwise, specify a port used for the server.

The next step is to configure the server instance itself.

Step 2. In SQL Server Management Studio, right-click the server instance and choose Properties > Connections. Then allow remote connections to your server:



Connections

Maximum number of concurrent connections (0 = unlimited): 0

[ ] Use query governor to prevent long-running queries
0

Default connection options:
[ ] implicit transactions
[ ] cursor close on commit
[ ] ansi warnings
[ ] ansi padding
[ ] ANSI NULLS
[ ] arithmetic abort
[ ] arithmetic ignore
[ ] quoted identifier
[ ] no count
[ ] ANSI NULL Default On

Remote server connections

[✓] Allow remote connections to this server

Remote query timeout (in seconds, 0 = no timeout): 600

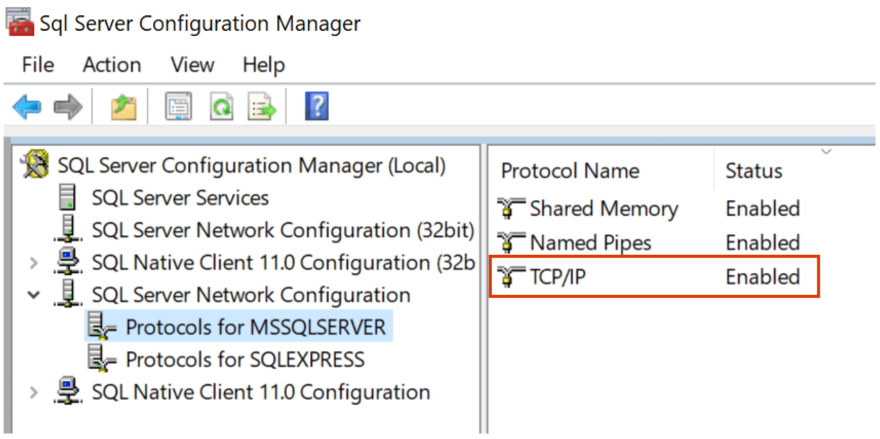
[ ] Require distributed transactions for server-to-server communication

[● Configured values] Running values

Server:
DESKTOP-KOJP18S

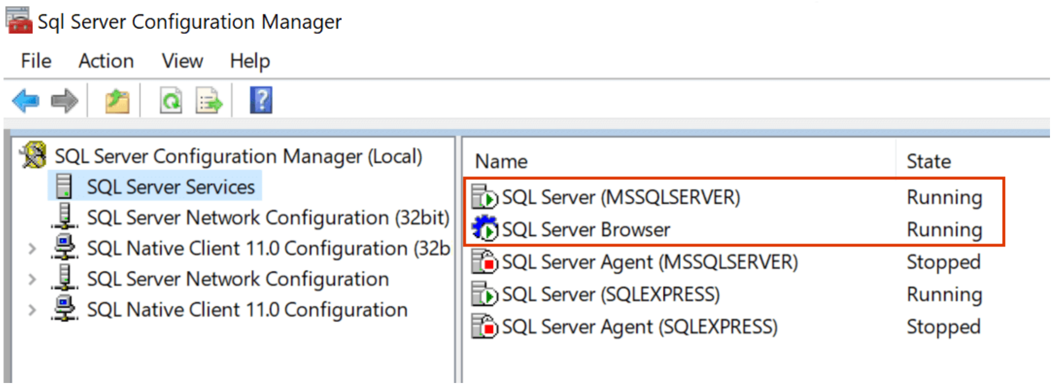
Connection:
DESKTOP-KOJP18S\twitt

Step 3. Open SQL Server Configuration Manager and enable TCP/IP for the server instance:


Sql Server Configuration Manager
File Action View Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration (32bit)
SQL Native Client 11.0 Configuration (32b
SQL Server Network Configuration
Protocols for MSSQLSERVER
Protocols for SQLEXPRESS
SQL Native Client 11.0 Configuration

Protocol Name	Status
Shared Memory	Enabled
Named Pipes	Enabled
TCP/IP	Enabled

Step 4. This step also requires SQL Server Configuration Manager. In the SQL Server Services tab, run the needed SQL Server instance and SQL Server Browser:


Sql Server Configuration Manager
File Action View Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration (32bit)
SQL Native Client 11.0 Configuration (32b
SQL Server Network Configuration
SQL Native Client 11.0 Configuration

Name	State
SQL Server (MSSQLSERVER)	Running
SQL Server Browser	Running
SQL Server Agent (MSSQLSERVER)	Stopped
SQL Server (SQLEXPRESS)	Running
SQL Server Agent (SQLEXPRESS)	Stopped

For details on how to run SQL Server Browser, see this Microsoft's guide (https://docs.microsoft.com/en-us/dynamics-nav/how-to--start-sql-browser-service).

Step 5. Open the 1433 port for TCP in the Windows firewall. Refer to the Microsoft's guide (https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-a-windows-firewall-for-database-engine-access?view=sql-server-ver15#to-open-a-port-in-the-windows-firewall-for-tcp-access) for instructions. Note that for named SQL Server instances, the 1434 port should be opened for UDP.

Step 6. Run the Data Server:

on Windows

```
flexmonster-data-server.exe
```

on macOS and Ubuntu/Linux

```
./flexmonster-data-server
```

Now the Data Server should successfully connect to your Microsoft SQL Server database.

What’s next?

You may be interested in the following articles:

- Flexmonster Data Server configurations reference (/doc/configurations-reference/)
- Flexmonster Data Server security guide (/doc/security-and-authorization-guide/)
- How to configure which data subset is shown (/doc/slice/)
- How to specify functionality available to customers (/doc/options/)
- How to define number formatting (/doc/number-formatting/)

4.4.4. Connecting to a PostgreSQL database using Flexmonster Data Server

This tutorial describes how to connect to a PostgreSQL database using Flexmonster Data Server (/doc/intro-to-flexmonster-data-server/) – a special server developed by Flexmonster. This server communicates with the client using the custom data source API (/doc/introduction-to-custom-data-source-api/) – our custom communication protocol allowing you to retrieve already aggregated data from a server to Flexmonster Pivot.

Prerequisites

To download the Data Server, you will need Flexmonster CLI (https://www.flexmonster.com/doc/cli-overview/) — a command-line interface tool for Flexmonster. If needed, install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

After that, a new flexmonster command will be available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

Now follow the steps below to connect to PostgreSQL using the Data Server.

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Download Flexmonster Data Server

The previous step demonstrated how to configure Flexmonster Pivot. Now it’s time to set up Flexmonster Data Server.

Get the Data Server with the following CLI command:

```
flexmonster add fds
```

The flexmonster add fds command will download the .zip archive with Flexmonster Data Server and automatically unpack the files in the current folder. As a result, the flexmonster-data-server/ folder will appear in your working directory.

The Data Server can be configured in the flexmonster-data-server/flexmonster-config.json file. To learn more about all the available configurations, see the configurations reference (/doc/configurations-reference/).

To set configurations needed for a PostgreSQL database, follow the steps below.

Step 3. Configure the data source type

Set the "Type" property in flexmonster-config.json to "database". It should be done as follows:

```
"DataSources": [
    {
        "Type": "database"
    }
],
```

Step 4. Configure the database type

Specify the name of the database you want to use by adding the "DatabaseType" configuration to the flexmonster-config.json file. In this case, the configuration value should be "postgresql":

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "postgresql"
    }
]
```

Step 5. Configure the connection with the database

To enable the server to fetch data from your database, you have to provide the connection string to the database. The connection string should be added to flexmonster-config.json as follows:

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "postgresql",
        "ConnectionString":
 "Server=localhost;Port=5432;Uid=root;Pwd=password;Database=database_name"
    }
]
```

Have a look at the examples of different connection strings for PostgreSQL (https://www.connectionstrings.com/npgsql/).

Step 6. Define the data subset

In the flexmonster-config.json file, create an index for the subset of data you want Flexmonster Pivot to visualize. "Query" is an SQL query for the data. For example:

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "postgresql",
        "ConnectionString":
"Server=localhost;Port=5432;Uid=root;Pwd=password;Database=database_name",
        "Indexes": {
            "index_database": {
                "Query": "SELECT * FROM tablename"
            }
        }
    }
]
```

"index_database" is a dataset identifier. It will be used to configure the data source on the client side.

In a similar way, additional indexes can be specified:

```
"Indexes": {
    "index_database": {
        "Query": "SELECT * FROM tablename"
    },
    "another_index_database": {
        "Query": "SELECT column FROM tablename"
    }
}
```

Step 7. Run the server

To start the Data Server, run the following command in the console:

on Windows

```
flexmonster-data-server.exe
```

on macOS and Ubuntu/Linux

```
./flexmonster-data-server
```

As soon as you start Flexmonster Data Server, it automatically preloads the data specified in the "Indexes" property. Thus, when Flexmonster Pivot requests the data, the server responds with already preloaded data.

The Data Server keeps preloaded data in the server’s RAM, so the number of indexes you can specify is limited by the server’s RAM amount.

Step 8. Configure the report

On the client side, the report should be configured as follows:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:9500",
            index: "index-database"
        }
    }
});
```

index must match the name of the index defined in step 6 (e.g., "index_database").

When Flexmonster Pivot requests the data, Flexmonster Data Server server sends the response and then caches it. In case the component sends the same request once again, the server responds with the data from its cache. The Data Server clears the cache when restarted.

The Data Server’s cache has a memory limit. When the cache does not have enough memory for a new response, the Data Server server deletes one of the previously cached responses.

You can manage the cache size via the “CacheSizeLimit” property (https://www.flexmonster.com/doc/configurations-reference/#CacheSizeLimit).

If any errors appear when trying to connect to PostgreSQL, refer to the troubleshooting section (#troubleshooting).

Troubleshooting

Error: Cannot connect to the database using “Server=xxxx;Port=xxxx;Uid=xxxx;Pwd=xxxx;Database=xxxx”. Please check connection string

In case of connection to the secure PostgreSQL instance, some additional configurations are needed. Learn more details in this forum thread (/question/error-while-reading-from-stream/).

What’s next?

You may be interested in the following articles:

- Flexmonster Data Server configurations reference (/doc/configurations-reference/)

- Flexmonster Data Server security guide (/doc/security-and-authorization-guide/)
- How to configure which data subset is shown (/doc/slice/)
- How to specify functionality available to customers (/doc/options/)
- How to define number formatting (/doc/number-formatting/)

4.4.5. Connecting to an Oracle database using Flexmonster Data Server

This tutorial describes how to connect to an Oracle database using Flexmonster Data Server (/doc/intro-to-flexmonster-data-server/) – a special server developed by Flexmonster. This server communicates with the client using the custom data source API (/doc/introduction-to-custom-data-source-api/) – our custom communication protocol allowing you to retrieve already aggregated data from a server to Flexmonster Pivot.

Prerequisites

To download the Data Server, you will need Flexmonster CLI (https://www.flexmonster.com/doc/cli-overview/) — a command-line interface tool for Flexmonster. If needed, install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

After that, a new flexmonster command will be available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

Now follow the steps below to connect to Oracle using the Data Server.

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
```

```
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Download Flexmonster Data Server

The previous step demonstrated how to configure Flexmonster Pivot. Now it’s time to set up Flexmonster Data Server.

Get the Data Server with the following CLI command:

```
flexmonster add fds
```

The flexmonster add fds command will download the .zip archive with Flexmonster Data Server and automatically unpack the files in the current folder. As a result, the flexmonster-data-server/ folder will appear in your working directory.

The Data Server can be configured in the flexmonster-data-server/flexmonster-config.json file. To learn more about all the available configurations, see the configurations reference (/doc/configurations-reference/).

To set configurations needed for an Oracle database, follow the steps below.

Step 3. Configure the data source type

Set the "Type" property in flexmonster-config.json to "database". It should be done as follows:

```
"DataSources": [
    {
        "Type": "database"
    }
],
```

Step 4. Configure the database type

Specify the name of the database you want to use by adding the "DatabaseType" configuration to the flexmonster-config.json file. In this case, the configuration value should be "oracle":

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "oracle"
    }
]
```

Step 5. Configure the connection with the database

To enable the server to fetch data from your database, you have to provide the connection string to the database. The connection string should be added to flexmonster-config.json as follows:

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "oracle",
        "ConnectionString": "Data Source=ORCL;User Id=root;Password=password;"
    }
]
```

ORCL is a variable defined in the tnsnames.ora file (https://www.orafaq.com/wiki/Tnsnames.ora). The variable can be defined in the following way:

```
ORCL =
 (DESCRIPTION =
   (ADDRESS_LIST =
     (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
   )
   (CONNECT_DATA =
     (SERVICE_NAME = orcl)
   )
```

```
  )
```

Have a look at the examples of different connection strings for Oracle (https://www.connectionstrings.com/oracle-data-provider-for-net-odp-net/).

Step 6. Define the data subset

In the flexmonster-config.json file, create an index for the subset of data you want Flexmonster Pivot to visualize. "Query" is an SQL query for the data. For example:

```
"DataSources": [
    {
        "Type": "database",
        "DatabaseType": "oracle",
        "ConnectionString": "Data Source=ORCL;User Id=root;Password=password;",
        "Indexes": {
            "index_database": {
                "Query": "SELECT * FROM tablename"
            }
        }
    }
]
```

"index_database" is a dataset identifier. It will be used to configure the data source on the client side.

In a similar way, additional indexes can be specified:

```
"Indexes": {
    "index_database": {
        "Query": "SELECT * FROM tablename"
    },
    "another_index_database": {
        "Query": "SELECT column FROM tablename"
    }
}
```

Step 7. Run the server

To start the Data Server, run the following command in the console:

on Windows

```
flexmonster-data-server.exe
```

on macOS and Ubuntu/Linux

```
./flexmonster-data-server
```

As soon as you start Flexmonster Data Server, it automatically preloads the data specified in the "Indexes" property. Thus, when Flexmonster Pivot requests the data, the server responds with already preloaded data.

The Data Server keeps preloaded data in the server’s RAM, so the number of indexes you can specify is limited by the server’s RAM amount.

Step 8. Configure the report

On the client side, the report should be configured as follows:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:9500",
            index: "index-database"
        }
    }
});
```

index must match the name of the index defined in step 6 (e.g., "index_database").

When Flexmonster Pivot requests the data, Flexmonster Data Server server sends the response and then caches it. In case the component sends the same request once again, the server responds with the data from its cache. The Data Server clears the cache when restarted.

The Data Server’s cache has a memory limit. When the cache does not have enough memory for a new response, the Data Server server deletes one of the previously cached responses.

You can manage the cache size via the “CacheSizeLimit” property (https://www.flexmonster.com/doc/configurations-reference/#CacheSizeLimit).

What’s next?

You may be interested in the following articles:

- Flexmonster Data Server configurations reference (/doc/configurations-reference/)
- Flexmonster Data Server security guide (/doc/security-and-authorization-guide/)
- How to configure which data subset is shown (/doc/slice/)
- How to specify functionality available to customers (/doc/options/)

- How to define number formatting (/doc/number-formatting/)

4.5. Flexmonster Data Server

4.5.1. Introduction to Flexmonster Data Server

This section illustrates how to connect to a data source using Flexmonster Data Server.

Flexmonster Data Server is a special server-side tool implementing the custom data source API (https://www.flexmonster.com/doc/introduction-to-custom-data-source-api/). It is responsible for fetching data from a data source, processing, and aggregating it. Then the data is passed to Flexmonster Pivot in a ready-to-show format. The Data Server significantly reduces the time of data loading and enables analyzing large datasets.

Flexmonster Data Server supports the following data sources:

- CSV and JSON files
- MySQL databases
- Microsoft SQL Server databases
- PostgreSQL databases
- Oracle databases
- Microsoft Azure SQL databases

Advantages of Flexmonster Data Server:

- **Larger datasets**. Despite Flexmonster Pivot can connect to a CSV or JSON file directly, the browser's capability limits the size of a file (on average, up to 100 MB). Flexmonster Data Server can process files more than 1GB — the maximum file size depends on your RAM capacity and the number of unique members in the dataset.
- **Reduced browser memory usage.** Loading, processing, filtering, and aggregating of the data is made on the server side, so Flexmonster gets the ready-to-show data. Thus, the data is visualized faster, and the browser's resources are used more efficiently.
- **Built-in security configurations.** The Data Server allows establishing authorized access to data and managing security in a convenient way.
- **A convenient way of connecting to a database.** Though there are several ways of fetching data from a database, the speed and capacity of the Data Server make it the best way to load the data from a database.
- **Cross-platform solution**. Flexmonster Data Server is available for popular operating systems: Windows, Ubuntu/Linux, and macOS.

There are two ways to use Flexmonster Data Server:

- Using the console application (https://www.flexmonster.com/doc/getting-started-with-data-server/)
- Referencing the Data Server as a DLL (/doc/referring-data-server-as-dll/)

What's next?

You may be interested in the following articles:

- Getting started with the Data Server (/doc/getting-started-with-data-server/)
- Configurations reference (/doc/configurations-reference/)

- Data sources guide (/doc/data-sources-guide/)
- Security and authorization guide (/doc/security-and-authorization-guide/)

4.5.2. Getting started with Flexmonster Data Server

This walkthrough describes how to install Flexmonster Data Server. Flexmonster Pivot can be downloaded here (/download-page/).

Follow these guides to start using the Data Server:

- Installing Flexmonster Data Server (#install-data-server)
- Connecting to the Data Server (#connect-to-data-server)

Installing Flexmonster Data Server

For easy and smooth installation of Flexmonster Data Server, follow the steps below.

Prerequisites

Flexmonster Data Server can be installed with Flexmonster CLI (/doc/cli-overview/) — a command-line interface tool for Flexmonster. Install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

Now a new flexmonster command is available in the console. Learn more about Flexmonster CLI and its commands in our documentation (/doc/cli-overview/).

Step 1. Download

Flexmonster Data Server is available for the following operating systems: Windows (both 32-bit and 64-bit), Ubuntu/Linux, and macOS. Depending on your operating system, Flexmonster CLI will download the appropriate version of the Data Server.

To start the installation, run the following CLI command:

```
flexmonster add fds
```

Step 2. See the download package structure

The flexmonster add fds command will download the .zip archive with Flexmonster Data Server and automatically unpack the files in the current folder.

As a result, the flexmonster-data-server/ folder will appear in your working directory. It has the following structure:

- flexmonster-config.json – the Flexmonster Data Server configuration file. It contains a configured ready-to-use CSV data source "sample-index".
- flexmonster-data-server or flexmonster-data-server.exe for Windows – an executable version of

Flexmonster Data Server.
- flexmonster-setup-users or flexmonster-setup-users.exe for Windows – an additional utility for security and authorization management.
- sample-data/data.csv – the file with sample CSV data to create "sample-index" (see flexmonster-config.json).

Step 3. Run the Data Server

To start using Flexmonster Data Server, just run the executable file from the console:

on Windows

```
flexmonster-data-server.exe
```

on macOS and Ubuntu/Linux

```
./flexmonster-data-server
```

Now Flexmonster Data Server is up and running. To connect Flexmonster Pivot to the Data Server, follow the next guide.

Connecting to the Data Server

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Configure the report

To connect to Flexmonster Data Server, configure the report as follows:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:9500",
            index: "sample-index"
        }
    }
});
```

"sample-index" is the index defined in flexmonster-config.json for the sample-data/data.csv file.

Open the web page in the browser: the pivot table with the sample CSV data is shown.

What’s next?

You may be interested in the following articles:

- Data sources guide (https://www.flexmonster.com/doc/data-sources-guide/)
- Configurations reference (https://www.flexmonster.com/doc/configurations-reference/)
- Security and authorization guide (https://www.flexmonster.com/doc/security-and-authorization-guide/)

4.5.3. Configurations reference

This tutorial describes Flexmonster Data Server configuration. To install the Data Server, refer to Getting started with the Data Server (/doc/getting-started-with-data-server/).

Table of contents:

- Available configurations (#available-configurations)
- Setting configurations dynamically (#dynamic-config)
- Setting the port number from the command line (#port-from-console)
- Specifying a path to the configuration file (#path-to-config)
- Examples (#examples)

Available configurations

Flexmonster Data Server can be configured in the flexmonster-config.json file. It contains the following properties:

- "DataSources" – Array of objects. Allows configuring the data sources. Each object has the following properties:
    - "Type" – String. The type of the data source: "json", "csv", or "database".
    - "DatabaseType" (optional) – String. The type of the database: "mysql", "mssql", "postgresql", or "oracle". Only for "database" data source type.
    - "ConnectionString" (optional) – String. A connection string for the database. Only for "database" data source type.
    - "Indexes" – Object. Contains a list of datasets. Each dataset is represented by a "key": "value" pair, where "key" is a dataset name, and "value" is an object with the following properties:
        - "Path"(optional) – String. The path to the file with data. Only for "json" and "csv" data source types.
        - "Query" (optional) – String. The query to execute (e.g., "SELECT * FROM tablename"). Only for "database" data source type.
        - "Delimiter" (optional) – String. Defines the specific fields separator to split each CSV row. There is no need to define it if the CSV fields are separated by ,. This property is required only if another character separates fields. *Default value: ","*.

        - "DecimalSeparator" (optional) – String. Defines the specific character used to separate decimal parts of numbers. For example, to import CSV data with commas used to separate decimal parts of numbers (e.g., 3,14), set the "DecimalSeparator" property as ",". *Default value: "."*.
        - "ThousandSeparator" (optional) – String. Defines the specific character used to separate groups of digits in numbers. For example, to import CSV data with periods used to separate groups of digits in numbers (e.g., 1.000 for one thousand), set the "ThousandSeparator" property as ".". *Default value: ","*.
- "Security" (optional) – Object. Allows managing the data access security. It contains the following properties:
    - "Authorization" (optional) – Object. Has the following properties:
        - "Enabled" – Boolean. Indicates whether the “Basic Authorization” for Flexmonster Data Server is enabled. When set to true, the authorization is enabled. *Default value: false*.
    - "CORS" (optional) – Object. Allows configuring the cross-origin resource sharing for Flexmonster Data Server. It contains the following properties:
        - "AllowOrigin" (optional) – String. The origin from which the server accepts the requests. If "AllowOrigin" is set to "*", requests from all origins are accepted.
          Note that if authorization is enabled ("Enabled": true), "*" cannot be set as the origin. In this case, specific origins must be defined. Several origins must be defined as follows:

          ```
          "AllowOrigin": "http://localhost,https://example.com"
          ```

          Default value: "".*
    - "HSTS" (optional) — Object. Allows configuring the Strict-Transport-Security (HSTS (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security)) response header. Contains the following properties:
        - "MaxAge" — Number. Defines how long the browser remembers that the site should be accessed only via HTTPS. The "MaxAge" property is set in seconds.
        - "IncludeSubDomains" (optional) — Boolean. Defines whether the Strict-Transport-Security header applies to the site’s subdomains (true) or not (false). *Default value: false*.
        - "Preload" (optional) — Boolean. Defines whether the site is in the HSTS preload list (true) or not (false). For details refer to the MDN documentation about HSTS (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security#Preloading_Strict_Transport_Security). *Default value: false*.
    - "Headers" (optional) — Object. Allows adding response headers. This object consists of "key": "value" pairs, where "key" is a header name, and "value" is its value.
      We recommend adding response headers carefully and only if they are required for the project. Specifying an existing header with the wrong value can lead to runtime errors.
      Here is an example of how response headers can be specified:

      ```
      "Headers": {
          "Content-Security-Policy": "default-src 'self'",
          "X-Content-Type-Options": "nosniff",
          "X-Frame-Options": "SAMEORIGIN",
          "X-XSS-Protection": "1; mode=block"
      }
      ```

- "DataStorageOptions" (optional) – Object. Allows configuring options for data storage. It has the following properties:
    - "DataRefreshTime" (optional) – Number. Defines how often the data is reloaded from a file or a database. The refresh time is set in minutes. If the "DataRefreshTime" is not specified, the data will not be reloaded.
    - "CacheSizeLimit" (optional) – Number. The maximum number of cached server responses for every index defined in the "DataSources" property. When set to 0, the Data Server does not cache

the responses. *Default value: 100*.

- "Port" (optional) – String. The number of the port Flexmonster Data Server runs on. The port can also be set from the command line (#port-from-console). *Default value: "9500"*.
- "LoggerMinLevel" (optional) – String. Defines a minimum log level for the Data Server: "Info", "Warn", "Error", or "Fatal". Setting "LoggerMinLevel" to "Warn" or a higher level can improve the Data Server's performance. Learn more about these log levels in the Microsoft documentation (https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-5.0#log-level). "LoggerMinLevel" will not affect info-level logs that appear when the Data Server is started, or the data is reloaded.
  Default value: "Info".
- "HTTPS" (optional) – Object. Allows configuring the HTTPS protocol. It contains the following properties:
  - "Enable" (optional) – Boolean. Indicates whether the HTTPS protocol is enabled (true) or not (false). *Default value: false*.
  - "Certificate" (optional) – Path-Password Object|Subject-Store Object. Allows adding a certificate. By default, if the certificate is not specified, the generated development certificate will be used. The certificate can be added with either Path-Password Object or Subject-Store Object. Each of them has different properties.
    The Path-Password Object has the following properties:
    - "Path" – String. The URL to the certificate file. Note that Flexmonster Data Server supports only .pfx certificates.
    - "Password" – String. The password to access the certificate.

    The Subject-Store Object has the following properties:
    - "Subject" – String. The certificate subject name.
    - "Store" – String. The store from which the certificate is loaded.
    - "Location" (optional) – String. The location of the store from which to load the certificate. It can be either "CurrentUser" or "LocalMachine". *Default value: "CurrentUser".*
    - "AllowInvalid" (optional) – Boolean. Indicates whether to allow using invalid certificates (e.g., self-signed certificates). When set to true, invalid certificates are allowed to use. *Default value: false.*
  - "Protocols" (optional) – String. Establishes the HTTP protocols enabled on a connection endpoint or for the server. It can be one of the following values: "Http1", "Http2", and "Http1AndHttp2". *Default value: "Http1AndHttp2"*.

Setting configurations dynamically

To store the connection strings and other configurations more securely, you can set them dynamically as command-line arguments or environment variables.

Follow a short guide to set dynamic configurations:

Step 1. Assign the parameter to the needed configuration as follows:

```
"DataSources": [{
    "Type": "database",
    "DatabaseType": "postgresql",
    "ConnectionString": "${param}",
    "Indexes": {
        "index_database": {
            "Query": "SELECT * FROM ${param2}"
        }
    }
}],
```

param and param2 are the names of your parameters.

Step 2. Using the parameter’s name (e.g., param), pass your value to the Data Server:

As a command-line argument on Windows

```
flexmonster-data-server.exe param=connectionString param2=tableName
```

As a command-line argument on macOS and Ubuntu/Linux

```
./flexmonster-data-server param=connectionString param2=tableName
```

As an environment variable

Create an environment variable, where the variable name is your parameter’s name (e.g., param), and the variable value is a needed value (e.g., connectionString).

Setting the port number from the command line

To define a port for the Data Server from the command line, use the -p parameter:

On Windows

```
flexmonster-data-server.exe -p 7777
```

On macOS and Ubuntu/Linux

```
./flexmonster-data-server -p 7777
```

Note that if the "Port" property is specified in flexmonster-config.json, the port set from the console will not be applied.

Specifying a path to the configuration file

If flexmonster-config.json and the Data Server are located in different folders, you can specify the path to flexmonster-config.json when running the Data Server. It can be done using the -s parameter:

On Windows

```
flexmonster-data-server.exe -s <path_to_config>\flexmonster-config.json
```

On macOS and Ubuntu/Linux

```
./flexmonster-data-server -s <path_to_config>/flexmonster-config.json
```

Examples

Here is an example of configured flexmonster-config.json:

```
{
   "DataSources": [
     {
       "Type": "json",
       "Indexes": {
         "index_json": {
           "Path": "./data/data.json"
         }
       }
     },
     {
       "Type": "database",
       "DatabaseType": "mysql",
       "ConnectionString":
 "Server=localhost;Port=3333;Uid=root;Pwd=pwd;Database=TestDB",
       "Indexes": {
         "index_database": {
           "Query": "SELECT * FROM TestTable;"
         }
       }
     }
   ],
   "Security": {
     "Authorization": {
       "Enabled": false
     },
     "CORS": {
       "AllowOrigin": "*"
     },
     "HSTS": {
       "MaxAge": 31536000,
       "IncludeSubDomains": true
     },
     "Headers": {
       "Content-Security-Policy": "default-src 'self'",
       "X-Content-Type-Options": "nosniff",
       "X-Frame-Options": "SAMEORIGIN",
       "X-XSS-Protection": "1; mode=block"
     }
   },
   "DataStorageOptions": {
     "DataRefreshTime": 60,
     "CacheSizeLimit": 150
   },
   "Port": "9500",
   "HTTPS": {
     "Enabled": true,
     "Certificate": {
       "Subject": "localhost",
       "Store": "My"
     },
     "Protocols": "Http2"
   }
}
```

What’s next?

You may be interested in the following articles:

- Data sources guide (/doc/data-sources-guide)
- Security and authorization guide (/doc/security-and-authorization-guide/)

4.5.4. Data sources guide

This tutorial describes how to connect Flexmonster Data Server to your data. To learn more about the Data Server’s configurations, refer to our configurations guide (/doc/configurations-reference/).

Server configurations vary depending on the data source type:

- Connecting to JSON (#json)
- Connecting to CSV (#csv)
- Connecting to databases (#database)

Connecting to JSON

To connect to JSON with Flexmonster Data Server, specify the "Type" and "Indexes" properties in the flexmonster-config.json file. For example:

```
"DataSources": [
    {
        "Type": "json",
        "Indexes": {
            "index_json": {
                "Path": "../data/data.json"
            }
        }
    }
],
```

"index_json" is a dataset identifier. It will be used to configure the data source on the client side.

Learn more about connecting to JSON with the Data Server in our detailed guide: Connecting to JSON using Flexmonster Data Server (/doc/pivot-table-connect-to-json/).

To start Flexmonster Data Server, refer to the Run Flexmonster Data Server (#run-server) section.

To see how the connection with Flexmonster Data Server is configured in the component, refer to the Configuring the report (#client) section.

Connecting to CSV

To connect to CSV with Flexmonster Data Server, specify the "Type" and "Indexes" properties in the flexmonster-config.json file. For example:

```
"DataSources": [
    {
        "Type": "csv",
        "Indexes": {
            "index_csv": {
                "Path": "../data/data.csv"
            }
        }
    }
],
```

"index_csv" is a dataset identifier. It will be used to configure the data source on the client side.

Learn more about connecting to CSV with the Data Server in our detailed guide: Connecting to CSV using Flexmonster Data Server (/doc/pivot-table-connect-to-csv/).

To start Flexmonster Data Server, refer to the Run Flexmonster Data Server (https://www.flexmonster.com/console/post.php?post=27539&action=edit#run-server) section.

To see how the connection with Flexmonster Data Server is configured in the component, refer to the Configuring the report (#client) section.

Connecting to databases

Flexmonster Data Server supports the following databases: MySQL, Microsoft SQL Server, PostgreSQL, Oracle, and Microsoft Azure SQL.

To connect to a database with Flexmonster Data Server, specify the "Type", "DatabaseType", "ConnectionString", and "Indexes" properties in the flexmonster-config.json file. For example:

```
{
    "DataSources": [
        {
            "Type": "database",
            "DatabaseType": "mysql"
            "ConnectionString": "Server=localhost;Port=3306;Uid=root;Pwd=password;Da
tabase=database_name",
            "Indexes": {
                "index_database": {
                    "Query": "SELECT * FROM tablename"
                }
            }
        }
    ]
}
```

"index_database" is a dataset identifier. It will be used to configure the data source on the client side.

Learn more about connecting to a database with the Data Server in our detailed guides:

- Connecting to a MySQL database (/doc/connect-to-mysql-database/)
- Connecting to a Microsoft SQL Server database (/doc/connect-to-mssql-database/) (works for Microsoft Azure SQL as well)
- Connecting to a PostgreSQL database (/doc/connect-to-postgresql-database/)
- Connecting to an Oracle database (/doc/connect-to-oracle-database/)

To start Flexmonster Data Server, refer to the Run Flexmonster Data Server (https://www.flexmonster.com/console/post.php?post=27539&action=edit#run-server) section.

To see how the connection with Flexmonster Data Server is configured in the component, refer to the Configuring the report (#client) section.

Run Flexmonster Data Server

To start Flexmonster Data Server, run the following command in the console:

on Windows

```
flexmonster-data-server.exe
```

on macOS and Ubuntu/Linux

```
./flexmonster-data-server
```

As soon as you start Flexmonster Data Server, it automatically preloads the data specified in the "Indexes" property. Thus, when Flexmonster Pivot requests the data, the server responds with already preloaded data.

The Data Server keeps preloaded data in the server’s RAM, so the number of indexes you can specify is limited by the server’s RAM amount.

Configuring the report

On the client side, the report should be configured as follows:

```
new Flexmonster({
    container: "pivotContainer",
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:9500",
            index: "index-json"
        }
    }
});
```

index must match the name of the index defined when configuring Flexmonster Data Server (e.g., "index_json").

When Flexmonster Pivot requests the data, Flexmonster Data Server server sends the response and then caches it. In case the component sends the same request once again, the server responds with the data from its cache. The Data Server clears the cache when restarted.

The Data Server’s cache has a memory limit. When the cache does not have enough memory for a new response, the Data Server server deletes one of the previously cached responses.

You can manage the cache size via the “CacheSizeLimit” property (https://www.flexmonster.com/doc/configurations-reference/#CacheSizeLimit).

What’s next?

- Configurations reference (https://www.flexmonster.com/doc/configurations-reference/)
- Security and authorization guide (https://www.flexmonster.com/doc/security-and-authorization-guide/)
- Connecting to JSON using Flexmonster Data Server (/doc/pivot-table-connect-to-json/)
- Connecting to CSV using Flexmonster Data Server (/doc/pivot-table-connect-to-csv/)
- Connecting to SQL databases using Flexmonster Data Server (/doc/connect-to-relational-database/)

4.5.5. Security and authorization guide

This tutorial describes how to configure the data access security in Flexmonster Data Server. To connect the Data Server to your data, refer to the Data sources guide (/doc/data-sources-guide/).

Flexmonster Data Server supports different essential security configurations, such as built-in basic authorization and HTTPS. To learn more about security configurations in the Data Server, see the following guides:

- Built-in basic authorization (#basic)
- Configure the HTTPS protocol (#https)
- Reverse proxy authorization (#reverse-proxy)
- Custom authorization and role-based permissions (#custom-auth)
- Secure configuration setting (#secure-configuration)

Built-in basic authorization

By default, Flexmonster Data Server is accessible to anyone who can reach its host. Using the built-in basic authorization, you can restrict access to Flexmonster Data Server.

Step 1. Create a user

The flexmonster-setup-users utility allows creating new users and managing them. Run the following command in the console to create a new user:

on Windows

```
flexmonster-setup-users.exe add <username>
```

on macOS and Ubuntu/Linux

```
./flexmonster-setup-users add <username>
```

Here, <username> is the name of the created user.

Then you will be prompted to create and confirm the password for the user.

With the flexmonster-setup-users utility, it is possible to see all created users, change the password for a user, and delete a user. Run the following command in the console to learn more about users management:

on Windows

```
flexmonster-setup-users.exe --help
```

on macOS and Ubuntu/Linux

```
./flexmonster-setup-users --help
```

Step 2. Enable authorization

In the flexmonster-config.json file, enable the authorization by setting the "Enabled" property of the "Authorization" object to true:

```
"Security" : {
    "Authorization": {
        "Enabled": true
    },
    ...
}
```

Step 3. Configure CORS

Basic Authorization requires certain origins to be defined in the Access-Control-Allow-Origin header. Origin is a domain that sends requests to Flexmonster Data Server (e.g., http://localhost:8080 or https://example.com).

To allow the origin to send requests to the Data Server, specify the "AllowOrigin" property in the flexmonster-config.json file:

```
"Security" : {
    ...
    "CORS": {
        "AllowOrigin": "http://localhost:8080"
    }
}
```

Several origins must be defined as follows:

```
"AllowOrigin": "http://localhost:8080,https://example.com"
```

Step 4. Configure credentials on the client side

In this step, credentials are configured in Flexmonster Pivot. There are two ways to configure credentials:

1. Use the withCredentials property:

   ```
   {
       dataSource: {
           type: "api",
           url: "http://localhost:9500",
           index: "data",
           withCredentials: true
       }
   }
   ```

   In this case, you need to enter your login and password when first connecting to Flexmonster Data Server.

2. Add a request header with basic authentication. The header should be added in the following way:

   ```
   {
       dataSource: {
           type: "api",
           url: "http://localhost:9500",
           index: "data",
           requestHeaders: {
               "Authorization": "Basic QWxhZGRpbjpPcGVuU2VzYW1l"
           }
       }
   }
   ```

   The header should be specified in the standard for basic authentication format (https://en.wikipedia.org/wiki/Basic_access_authentication#Client_side).
   In this case, the authorization will be automatic, and the browser will not ask for the login and password.

Configure the HTTPS protocol

All data sent by HTTP is not encrypted and can be inspected. To make the Data Server more secure, we added an option to enable the HTTPS protocol. Follow the steps below to configure a secure HTTPS connection.

Step 1. Enable the HTTPS protocol

To enable the HTTPS protocol, set the "Enabled" property of the "HTTPS" object to true in the flexmonster-config.json file:

```
"HTTPS": {
    "Enabled" : true
}
```

Step 2. (optional) Add a certificate

There is an option to add an SSL/TLS certificate. The certificate can be added with either Path-Password Object or Subject-Store Object. Each of them has different properties.

To add the certificate with the Path-Password Object, specify the path to the certificate and the password required to access it:

```
"HTTPS": {
    "Enabled": true,
    "Certificate": {
        "Path": "sampleCert.pfx",
        "Password": "samplePassword"
    }
}
```

Flexmonster Data Server supports only .pfx certificates.

To add the certificate with the Subject-Store Object, the following properties should be specified:

1. In the "Certificate" object, specify the certificate subject name and the certificate store from which to load the certificate:

   ```
   "HTTPS": {
       "Enabled": true,
       "Certificate": {
           "Subject": "localhost",
           "Store": "My"
       }
   }
   ```

2. (optional) Specify the location of the store from which to load the certificate. Skip this step if the needed location is "CurrentUser", since the default value of the location is "CurrentUser". Otherwise, set the "Location" property to "LocalMachine":

   ```
   "HTTPS": {
       "Enabled": true,
       "Certificate": {
           "Subject": "localhost",
           "Store": "My",
           "Location": "LocalMachine"
       }
   }
   ```

3. (optional) To allow using invalid certificates, such as self-signed certificates, set the "AllowInvalid" property to true:

   ```
   "HTTPS": {
       "Enabled": true,
       "Certificate": {
           "Subject": "localhost",
           "Store": "My",
           "Location": "LocalMachine",
           "AllowInvalid": true
       }
   }
   ```

Step 3. (optional) Configure the protocols

The "Protocols" property establishes the HTTP protocols enabled on a connection endpoint or for the server. The "Protocols" property can be one of the following values: "Http1", "Http2", and "Http1AndHttp2". For example:

```
"HTTPS": {
    "Enabled": true,
    "Certificate": {
        "Path": "sampleCert.pfx",
        "Password": "samplePassword"
    },
    "Protocols": "Http2"
}
```

Step 4. (optional) Configure HSTS

The Strict-Transport-Security (HSTS (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security#Description)) response header tells browsers that the site only accepts a connection through the HTTPS protocol. This makes the site usage more secure.

Configure HSTS for Flexmonster Data Server either via the "HSTS" property or via the "Headers" property.

Via the "HSTS" property

If HSTS is configured via the "HSTS" property, it will be automatically added to all the Data Server’s responses:

```
"Security" : {
    ...
    "HSTS": {
        "MaxAge": 31536000,
        "IncludeSubDomains": true
    }
}
```

Via the "Headers" property

If HSTS is configured via the "Headers" property, it will be returned only with a response to XHR:

```
"Security" : {
    ...
    "Headers": {
        "Strict-Transport-Security": "max-age=31536000; includeSubDomains"
    }
}
```

Learn more about the directives of HSTS in the MDN documentation (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security#Directives).

Restart the Data Server to apply the configurations. Now, the HTTPS protocol will be used instead of HTTP.

Reverse proxy authorization

If you need to use your own authorization mechanism, you can restrict the public access to Flexmonster Data Server and enable access to it through the reverse proxy. This approach requires implementing the proxy, which is responsible for the data access control. The proxy will decide which requests should be accepted and passed to the Data Server, and which requests should be declined.

The proxy has to implement the custom data source API (/doc/introduction-to-custom-data-source-api/) to handle requests from Flexmonster Pivot. Then the proxy will be able to redirect Flexmonster Pivot's requests to the Data Server. To see the full list of requests sent by Flexmonster Pivot, refer to our documentation (/api/all-requests/).

Custom authorization and role-based permissions

Role-based access is supported when using Flexmonster Data Server as a DLL. Flexmonster.DataServer.Core.dll allows performing server-side filtering, so it becomes possible to show different subsets of the data to different user groups.

To demonstrate the usage of server-side filtering for role-based permissions, we created an ASP.NET application with a custom server using Flexmonster.DataServer.Core.dll (https://github.com/flexmonster/flexmonster-data-server-dll). The GitHub repository contains a solution file DemoDataServerCore.sln, so the sample can be opened and launched via Visual Studio.

To start the sample application from the console, run the following commands:

```
cd DemoDataServerCore
dotnet restore
dotnet run
```

To see the result, open http://localhost:5000/ in the browser.

On the page, there is the pivot table and the drop-down menu. Select a role from the menu to see how it affects the data shown in Flexmonster.

To see how the server-side filtering is implemented in the sample server, refer to the FlexmonsterAPIController.cs (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/Controllers/FlexmonsterAPIController.cs) file.

To learn more about the server filter, see the Implementing the server filter (/doc/implementing-server-filter) guide.

Secure configuration setting

To store connection strings and other configurations more securely, set them dynamically as command-line arguments or environment variables. For details on passing dynamic configurations to the Data Server, see the configurations reference (/doc/configurations-reference/#!dynamic-config).

What's next?

You may be interested in the following articles:

- Data sources guide (https://www.flexmonster.com/doc/data-sources-guide)
- Configurations reference (https://www.flexmonster.com/doc/configurations-reference/)
- Implementing the custom data source API (/doc/implement-custom-data-source-api/)

4.5.6. The Data Server as a DLL

4.5.6.1. Getting started with the Data Server as a DLL

Overview

Flexmonster Data Server can be embedded into your project as a DLL – a separate flexible module. The advantages of using a DLL instead of the console application are the following:

- No need to launch the .exe file.
- Easy software updates.
- The ability to customize the Data Server.

For a quick start with Flexmonster Data Server as a DLL, have a look at the sample .NET Core application (#usage-example) with the Data Server embedded.

To embed Flexmonster Data Server into an existing or new .NET Core application, follow this guide: Referencing the Data Server as a DLL (/doc/referring-data-server-as-dll/).

Usage example

Flexmonster.DataServer.Core.dll is a flexible solution that can be used in many ways. To demonstrate how the Data Server can be used, we created an ASP.NET Core application with a custom server using Flexmonster.DataServer.Core.dll (https://github.com/flexmonster/flexmonster-data-server-dll).

The application repository contains a solution file DemoDataServerCore.sln, so the sample can be opened and launched with Visual Studio.

Prerequisites

To run the sample project, you need Microsoft .NET Core 3.0 or higher. Get it here (https://dotnet.microsoft.com/download) if it’s not already installed on your machine.

Run the sample project

To start the sample application from the console, run the following commands:

```
cd DemoDataServerCore
dotnet restore
dotnet run
```

To see the result, open http://localhost:5000/ in the browser.

Use cases

With Flexmonster.DataServer.Core.dll, you can restrict access to the data to certain roles using the server filter. On the http://localhost:5000/ page, there is a pivot table and a drop-down menu. Select a role from the menu to see how it affects the data shown in Flexmonster Pivot.

To see how the server-side filtering is implemented in the sample server, refer to the FlexmonsterAPIController.cs (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/Controllers/FlexmonsterAPIController.cs) file.

To learn more about the server filter, read through the Implementing the server filter (/doc/implementing-server-filter) guide.

This custom server also demonstrates how to load data from a custom data source. To work with a custom data source, implement an appropriate parser for the data. An example of a custom parser is shown in this CustomParser.cs (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/CustomParser.cs) file.

In the sample project, the data shown on the grid is from the custom parser, but "csv", "json", and "database" parser types can also be used. To make the server use multiple parsers, add a new object to the "DataSources" property in the appsettings.json file. For example:

```
"DataSources": [
  {
    "Type": "custom",
    "Indexes": {
      "custom-index": null
    }
  },
  {
    "Type": "json",
    "Indexes": {
      "json-index": {
        "Path": "data.json"
    }
  }
],
```

To learn more about the custom parser, see the Implementing the custom parser (/doc/implementing-custom-parser) guide.

What’s next?

You may be interested in the following articles:

- Referencing the Data Server as a DLL (/doc/referring-data-server-as-dll/)
- Implementing the server filter (https://www.flexmonster.com/doc/implementing-server-filter)
- Implementing the custom parser (https://www.flexmonster.com/doc/implementing-custom-parser)
- Configurations reference (/doc/dll-configurations-reference/)

4.5.6.2. Referencing the Data Server as a DLL

This guide describes the process of embedding Flexmonster Data Server in a .NET Core application. Follow the steps below to reference the Data Server as a DLL.

Prerequisites

In this tutorial, we use Visual Studio 2019 as an IDE. If you do not have Visual Studio installed, download it here (https://visualstudio.microsoft.com/vs/).

To use Flexmonster Data Server as a DLL, you need Microsoft .NET Core 3.0 or higher. Get it here (https://dotnet.microsoft.com/download) if it’s not already installed on your machine.

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
```

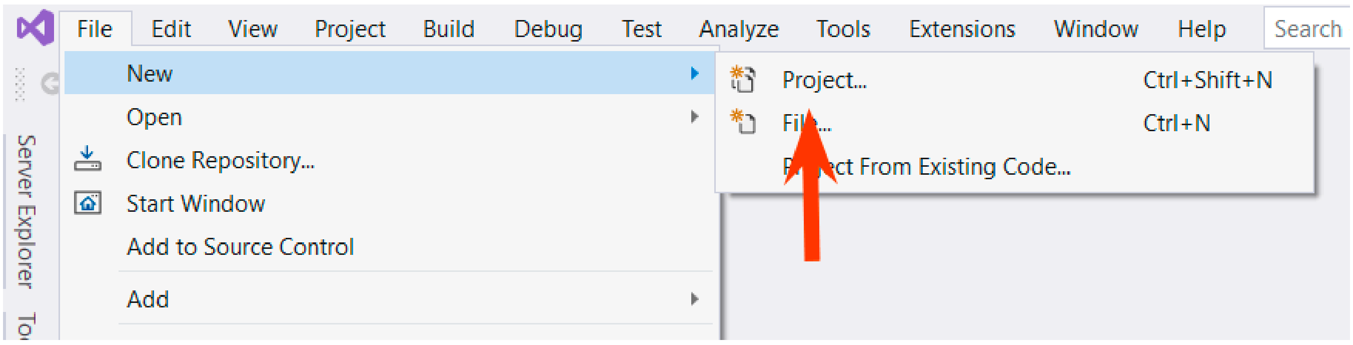
```
 ref="pivot"
 toolbar>
</Pivot>
```

Now it’s time to embed Flexmonster Data Server into your ASP.NET Core project.

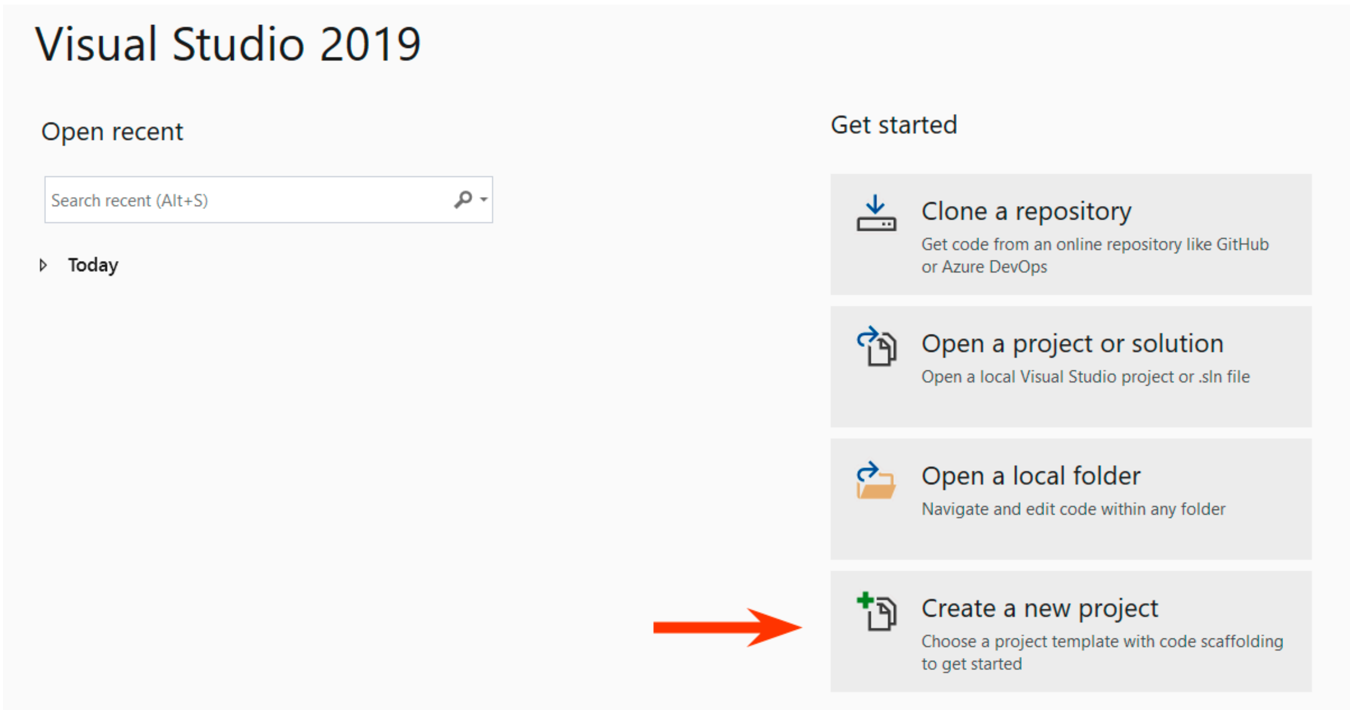
Step 2. Create a new ASP.NET Core web application

Skip this step if you already have an ASP.NET Core project. Otherwise, create a new project:

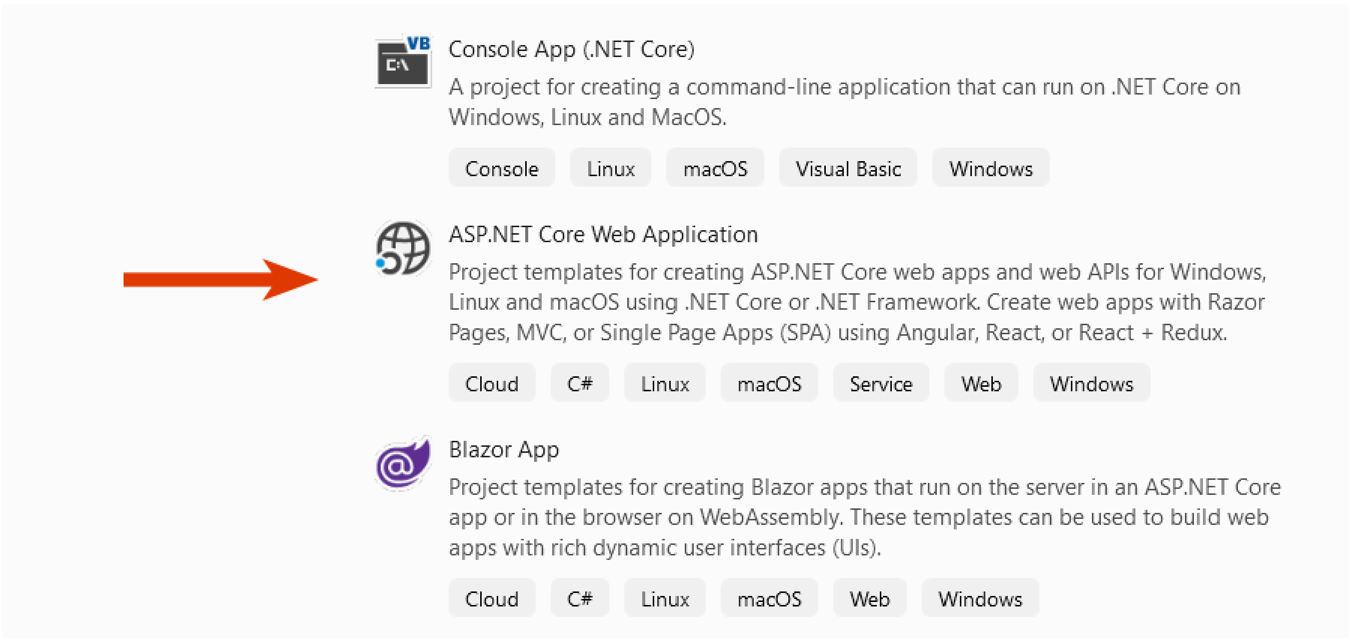
Via the File menu



Via the start window


Visual Studio 2019
Open recent
Search recent (Alt+S)
Today
Get started
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started

In the project templates menu, select the ASP.NET Core template:

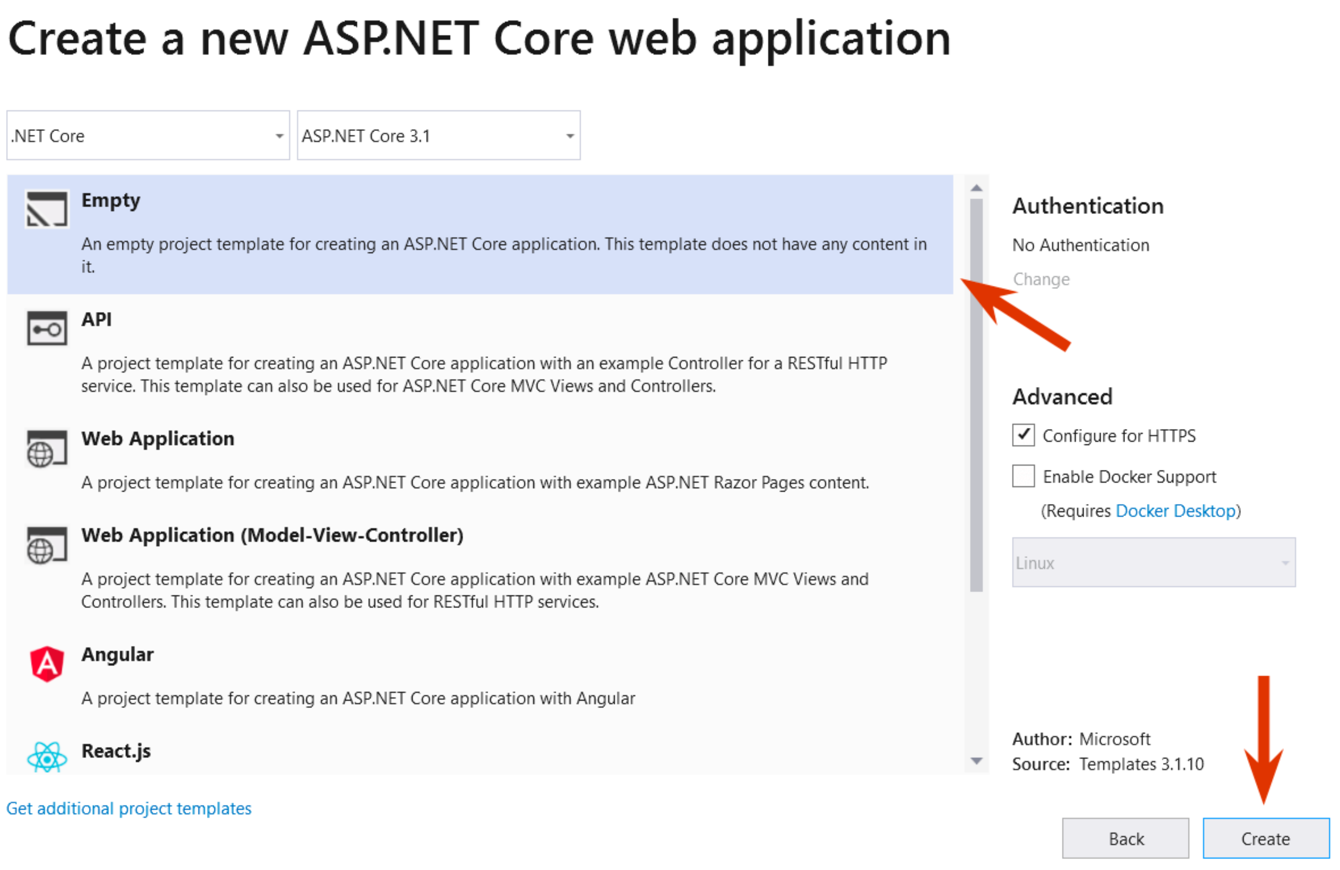


Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
Console | Linux | macOS | Visual Basic | Windows

ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.
Cloud | C# | Linux | macOS | Service | Web | Windows

Blazor App
Project templates for creating Blazor apps that run on the server in an ASP.NET Core app or in the browser on WebAssembly. These templates can be used to build web apps with rich dynamic user interfaces (UIs).
Cloud | C# | Linux | macOS | Web | Windows

Then, to simplify the server configuration process, create a project based on the API template:



Create a new ASP.NET Core web application

.NET Core | ASP.NET Core 3.1

Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Web Application
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

Web Application (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Angular
A project template for creating an ASP.NET Core application with Angular

React.js

Get additional project templates

Authentication
No Authentication
Change

Advanced
☑ Configure for HTTPS
☐ Enable Docker Support
(Requires Docker Desktop)
Linux

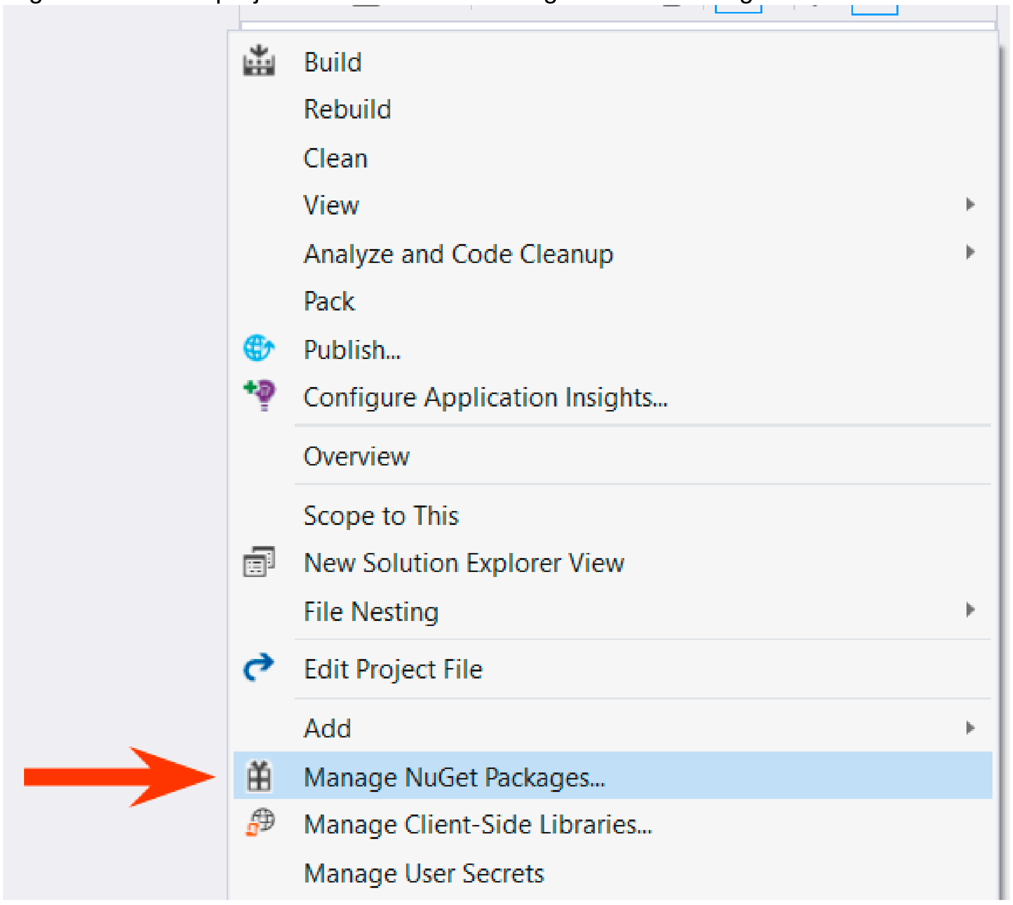
Author: Microsoft
Source: Templates 3.1.10

Back | Create

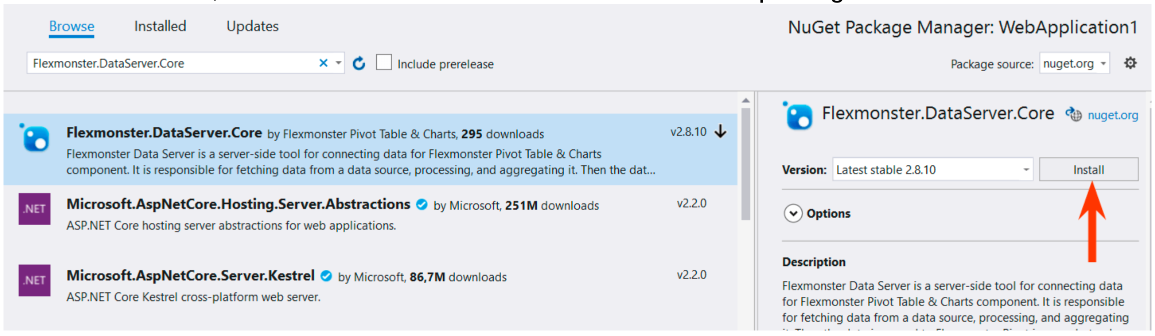
Step 3. Install Flexmonster.DataServer.Core.dll

Flexmonster.DataServer.Core.dll can be installed with NuGet – the Visual Studio's package manager:

1. Right-click on the project and select the Manage NuGet Packages item:



2. In the Browse tab, search for the Flexmonster.DataServer.Core package and install it:


Browse
Installed
Updates
NuGet Package Manager: WebApplication1
Flexmonster.DataServer.Core
Include prerelease
Package source: nuget.org
Install
Options

Flexmonster.DataServer.Core by Flexmonster Pivot Table & Charts, **295** downloads v2.8.10
Flexmonster Data Server is a server-side tool for connecting data for Flexmonster Pivot Table & Charts component. It is responsible for fetching data from a data source, processing, and aggregating it. Then the dat...

Microsoft.AspNetCore.Hosting.Server.Abstractions by Microsoft, **251M** downloads v2.2.0
ASP.NET Core hosting server abstractions for web applications.

Microsoft.AspNetCore.Server.Kestrel by Microsoft, **86,7M** downloads v2.2.0
ASP.NET Core Kestrel cross-platform web server.

Flexmonster.DataServer.Core nuget.org

Version: Latest stable 2.8.10

Description
Flexmonster Data Server is a server-side tool for connecting data for Flexmonster Pivot Table & Charts component. It is responsible for fetching data from a data source, processing, and aggregating

Another way to install Flexmonster.DataServer.Core.dll is by running the following command in the console:

```
dotnet add package Flexmonster.DataServer.Core
```

Step 4. Include Flexmonster.DataServer.Core in the Startup.cs file

Startup.cs is located in the project's root directory. Add the following line of code to the beginning of the file:

```
using Flexmonster.DataServer.Core;
```

Step 5. Register FlexmonsterAPIService in the dependency injection container

In the ConfigureServices() method of the Startup.cs file, add the following line of code:

```
services.AddFlexmonsterApi();
```

Step 6. Configure the data sources

For Flexmonster.DataServer.Core.dll, it is possible to configure data sources and different data storage options. The most convenient way of setting the Data Server configuration is to use the configuration file.

In the configuration file (e.g., appsettings.json), define the data sources to use. For example:

```
"DataSources":
[
  {
    "Type": "json",
    "Indexes": {
      "index-json": {
        "Path": "data.json"
      }
    }
  },
  //other data sources
],
```

To learn more about the available configurations for Flexmonster.DataServer.Core.dll, refer to the DLL configurations guide (/doc/dll-configurations-reference/).

Step 7. (optional) Set the data storage options

Data storage options include the data refresh time and the cache size limit. The data refresh time defines how often the data is reloaded from a file or a database. The cache size limit is the maximum number of cached server responses for every index specified in the "DataSources" property. You can set the configurations in the configuration file like this:

```
"DataStorageOptions": {
  "DataRefreshTime": 60,
  "CacheSizeLimit": 150
}
```

The data refresh time is set in minutes. If the refresh time is not defined, the data is not reloaded.

If the cache size limit is not specified, the number of cached responses is 100.

Step 8. Add a configuration object to the Startup.cs file

The IConfiguration object contains user configurations and is used to pass them to the Data Server. Create a variable with the IConfiguration type and initialize it in the constructor:

```
public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}

public IConfiguration Configuration { get; }
```

Step 9. Add the Data Server configurations

In the previous steps, we configured the Data Server in the configuration file. To apply this configuration, add the following line of code to the ConfigureServices() method of the Startup.cs file:

```
services.ConfigureFlexmonsterOptions(Configuration);
```

Now the ConfigureServices() method of the Startup.cs file should look similar to the following:

```
public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)
{
    // other services configurations
    services.ConfigureFlexmonsterOptions(Configuration);
    services.AddFlexmonsterApi();
    // other services configurations
}
```

Step 10. Implement the API controller

The API controller is responsible for handling the requests from Flexmonster Pivot. The implementation of the controller is described in the API controller guide (/doc/implementing-api-controller/).

After you implement the API controller, add the following lines of code to the ConfigureServices() and Configure() methods:

```
public void ConfigureServices(IServiceCollection services)
{
    // other configurations
    services.ConfigureFlexmonsterOptions(Configuration);
    services.AddFlexmonsterApi();
    services.AddControllers();
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    // other configurations
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}");
    });
}
```

Step 11. Enable cross-origin resource sharing (CORS)

Due to the same-origin policy (https://en.wikipedia.org/wiki/Same-origin_policy), the browser only allows requests that come from the same origin. Cross-origin resource sharing (CORS) allows web applications to make cross-domain requests.

If the ASP.NET Core server and the client share the same origin, skip this step. Otherwise, CORS must be enabled to allow Flexmonster Pivot to send requests to Flexmonster.DataServer.Core.dll:

```
public void ConfigureServices(IServiceCollection services)
{
    // other configurations
    services.ConfigureFlexmonsterOptions(Configuration);
    services.AddFlexmonsterApi();
    services.AddControllers();
    services.AddCors();
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    // other configurations
    app.UseCors(builder => {
        builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();
    });

    app.UseEndpoints(endpoints =>
     {
         endpoints.MapControllerRoute(
             name: "default",
             pattern: "{controller=Home}/{action=Index}");
     });
}
```

Find out how to enable CORS in ASP.NET Core here (https://docs.microsoft.com/en-us/aspnet/core/security/cors?view=aspnetcore-3.1).

Step 12. Run the project

The ASP.NET Core project can be run either using Visual Studio or the console. To run the project from the console, enter the following command:

```
dotnet run
```

Step 13. Configure the report

On the client side, configure the report as follows:

```
new Flexmonster({
    container: "pivotContainer",
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:5000/api", // replace with your url
            index: "index-json"
        }
    }
});
```

index must match the name of the index defined in step 6 (#configure-data-sources) (e.g., "index_json").

Now Flexmonster.DataServer.Core.dll is configured and ready to connect to Flexmonster Pivot. Open the pivot table in the browser to see the data.

What’s next?

You may be interested in the following articles:

- Implementing the custom parser (https://www.flexmonster.com/doc/implementing-custom-parser)
- Implementing the server filter (https://www.flexmonster.com/doc/implementing-server-filter/)
- DLL configurations reference (/doc/dll-configurations-reference/)

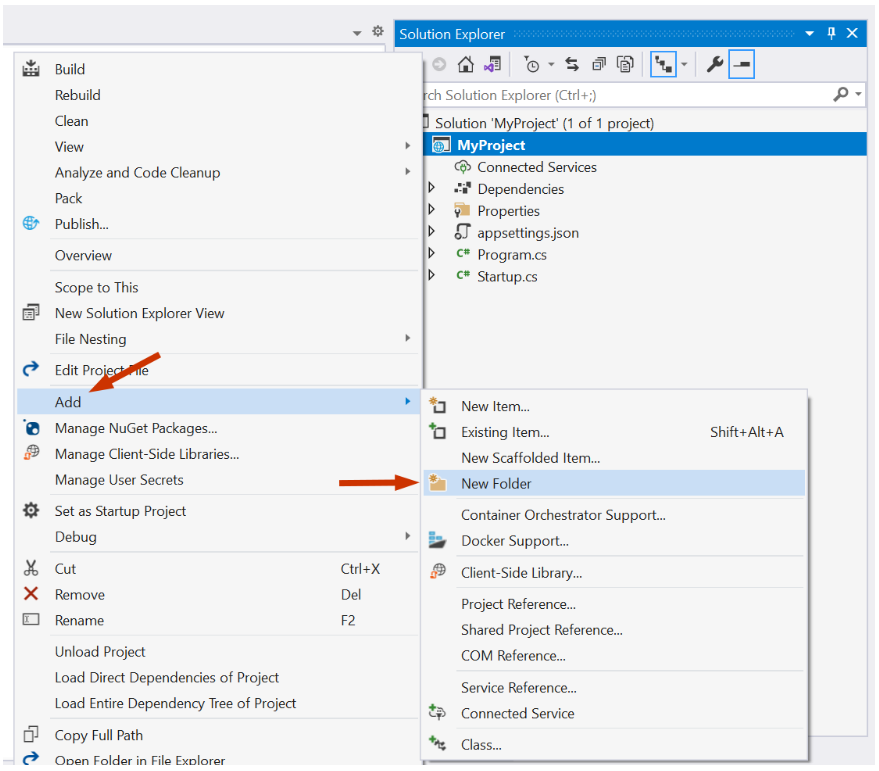
4.5.6.3. Implementing the API controller

The API controller is essential for communication between the server and Flexmonster Pivot. This guide describes how to implement the API controller for Flexmonster.DataServer.Core.dll.

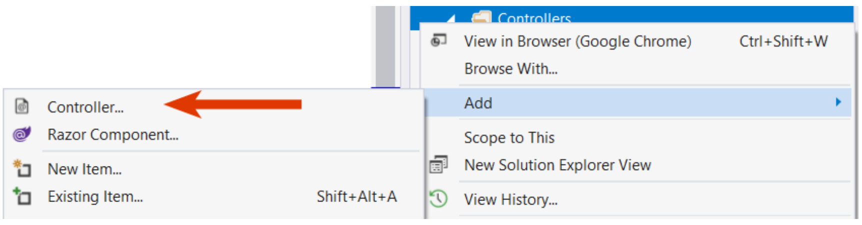
This tutorial relates to the guide Referencing the Data Server as a DLL (/doc/referencing-data-server-as-dll/) and should be completed as a part of it.

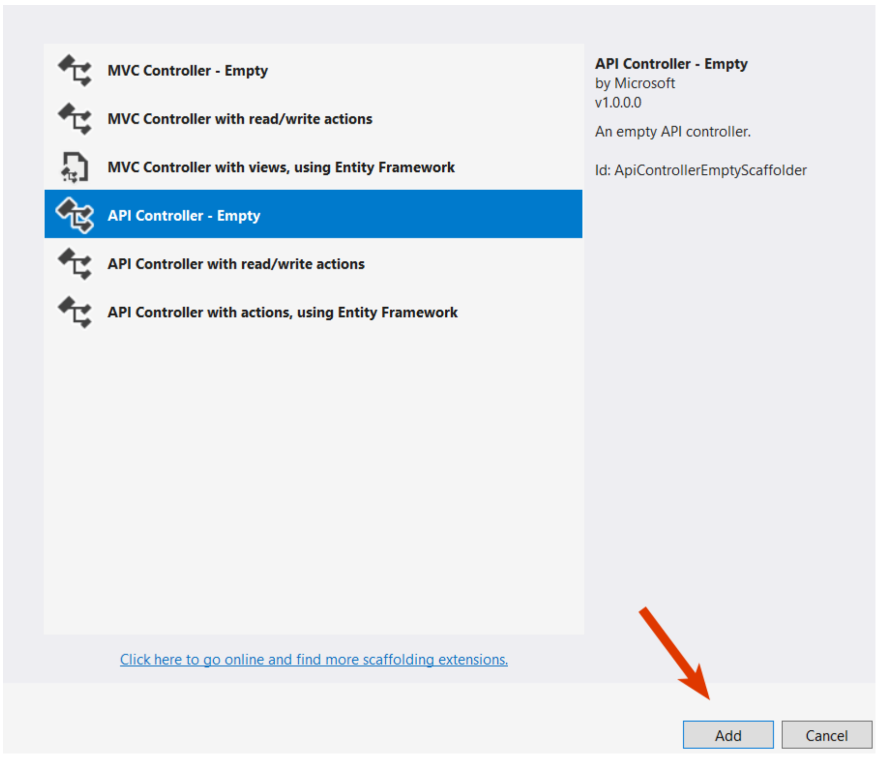
Step 1. Create a new API controller

If needed, create the Controllers folder inside the project:



Then create a new API controller class FlexmonsterAPIController inside this folder:


Controllers
View in Browser (Google Chrome) Ctrl+Shift+W
Browse With...
Add
Scope to This
New Solution Explorer View
View History...
Controller...
Razor Component...
New Item...
Existing Item... Shift+Alt+A


MVC Controller - Empty
MVC Controller with read/write actions
MVC Controller with views, using Entity Framework
API Controller - Empty
API Controller with read/write actions
API Controller with actions, using Entity Framework
API Controller - Empty
by Microsoft
v1.0.0.0
An empty API controller.
Id: ApiControllerEmptyScaffolder
Click here to go online and find more scaffolding extensions.
Add
Cancel

The code should be the following:

```
[Route("api")]
[ApiController]
public class FlexmonsterAPIController : ControllerBase
{
}
```

Step 2. Include Flexmonster.DataServer.Core in the controller

To use the DLL in the controller, add the following line of code to the beginning of the FlexmonsterAPIController.cs file:

```
using Flexmonster.DataServer.Core;
```

Step 3. Add the IApiService field to the controller

IApiService is a class from Flexmonster.DataServer.Core.dll that contains methods for handling custom data source API requests (/api/all-requests/). These methods enable getting fields, members, and aggregated data.

To use these methods, add the field of the IApiService type to the controller class:

```
private readonly IApiService _flexmonsterApiService;

public FlexmonsterAPIController(IApiService apiService)
{
    _flexmonsterApiService = apiService;
}
```

From here, you need to handle four POST requests: /handshake (/api/handshake-request/), /fields (/api/fields-request/), /members (/api/members-request/), and /select (/api/select-request-for-pivot-table/).

Step 4. Include Flexmonster.DataServer.Core.Models to the controller

The Flexmonster.DataServer.Core.Models class contains type definitions for requests and responses. To include it to the controller, add the following line of code to the beginning of the FlexmonsterAPIController.cs file:

```
using Flexmonster.DataServer.Core.Models;
```

Step 5. Handle the /handshake request

The /handshake request establishes a connection between the client and server sides. If the server sends the implemented version of the custom data source API in response to the /handshake request, the client can check whether the server and the client implement the same version of the custom data source API.

To receive notifications about version compatibility, respond to the /handshake request with the implemented version of the custom data source API:

```
const string API_VERSION = "2.8.5";

[Route("handshake")]
[HttpPost]
public IActionResult Handshake()
{
    object response = new { version = API_VERSION };
    return new JsonResult(response);
}
```

Handing the /handshake (https://www.flexmonster.com/api/handshake-request/) request is optional. Flexmonster Pivot will proceed to the next request even if the server does not handle it.

Step 6. Handle the /fields request

The first type of request that must be handled is the /fields (/api/fields-request/) request. To handle it, use the GetFields() method (/doc/controllers-methods-for-request-handling/#GetFields) of the IApiService class:

```
[Route("fields")]
[HttpPost]
public IActionResult PostFields([FromBody]FieldsRequest request)
{
    var response = _flexmonsterApiService.GetFields(request);
    return new JsonResult(response);
}
```

FieldsRequest is a predefined class from Flexmonster.DataServer.Core.Models. It describes the /fields (/api/fields-request/#request) request’s structure.

Step 7. Handle the /members request

The next required request to handle is the /members (/api/members-request/) request. It should be handled with the GetMembers() method (/doc/controllers-methods-for-request-handling/#GetMembers()) of the IApiService class. This can be done as follows:

```
[Route("members")]
[HttpPost]
public IActionResult PostMembers([FromBody]MembersRequest request)
{
    var response = _flexmonsterApiService.GetMembers(request);
    return new JsonResult(response);
}
```

MembersRequest is a predefined class from Flexmonster.DataServer.Core.Models. It describes the /members (/api/members-request/) request's structure.

Step 8. Handle the /select request

The last request that must be handled is the /select (/api/select-request-for-pivot-table/) request. To handle it, use the GetAggregatedData() method (/doc/controllers-methods-for-request-handling/#GetAggregatedData) of the IApiService class:

```
[Route("select")]
[HttpPost]
public IActionResult PostSelect([FromBody]SelectRequest request)
{
    var response = _flexmonsterApiService.GetAggregatedData(request);
    return new JsonResult(response);
}
```

SelectRequest is a predefined class from Flexmonster.DataServer.Core.Models. It describes the /select (/api/select-request-for-pivot-table/#request) request's structure.

After handling all the requests, the controller is ready for use. Now you can return to the Referencing Flexmonster Data Server as a DLL (/doc/referring-data-server-as-dll/#enable-cors) guide and complete it.

What's next?

You may be interested in the following articles:

- The controller's methods for request handling (https://www.flexmonster.com/doc/controllers-methods-for-request-handling/)
- Referencing the Data Server as a DLL (/doc/referring-data-server-as-dll/)
- Implementing the custom parser (/doc/implementing-custom-parser)
- Implementing the server filter (https://www.flexmonster.com/doc/implementing-server-filter/)
- DLL configurations reference (/doc/dll-configurations-reference/)

4.5.6.4. Implementing the server filter

The server filter can be used to limit which data is shown on the grid depending on a user's role. It allows filtering the data right on the application's back end, so Flexmonster Pivot receives and shows only the subset of the data that meets filtering conditions.

In this guide, we describe the concept of the server filter, its structure, and provide an example implementation:

- The general idea of the server filter (#general-idea)
- Example of a server filter (#server-filter-example)
- The server filter structure (#server-filter-structure)

The general idea of the server filter

The main advantage of the server filter is that it can be used as a data access management tool. Using the server filter, Flexmonster.DataServer.Core.dll will filter the data based on the user role received from Flexmonster Pivot.

Here is a possible server-side filtering implementation:

1. When Flexmonster Pivot sends a request for data to the server, the user’s role can be included in the request headers with customizeAPIRequest (/api/customizeapirequest/). This method allows editing the headers before the request is sent to the server.
2. After the Data Server accepts the request, the request headers can be read. Depending on the received role, the DLL can apply the appropriate filter to the data.
3. The resulting subset of data is sent to Flexmonster Pivot. The component gets the dataset specific to the current user.

To see how the server filter can be implemented, refer to the Example section (#server-filter-example).

Example of a server filter

To simplify the server filter’s implementation, we designed a sample ASP.NET Core application (https://github.com/flexmonster/flexmonster-data-server-dll) using Flexmonster.DataServer.Core.dll. This application contains an example server filter, which can be viewed in the FlexmonsterAPIController.cs (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/Controllers/FlexmonsterAPIController.cs#L54) file.

To create a new ASP.NET Core application with the Data Server or to embed the DLL in an existing project, refer to the guide on Flexmonster.DataServer.Core.dll (/doc/referring-data-server-as-dll/).

Defining user roles on the server side

In the sample, the data is filtered based on the user’s region token (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/Controllers/FlexmonsterAPIController.cs#L14) sent by the component using request headers (/api/customizeapirequest/#request-headers). A dictionary defines which subset of data should correlate to the received token. The dictionary contains all possible region tokens and field members corresponding to them:

```
private static Dictionary<string, List<object>> _userPermissions =
new Dictionary<string, List<object>>()
{
    /* a user with the "AdminToken" token will see
    the report on all the countries */
    {"AdminToken", null },

    /* a user with the "EuropeToken" token will see
    details about Germany and France */
    {"EuropeToken", new List(){ "Germany","France" } },

    /* a user with the "AmericaToken" token will see
    highlights about USA and Canada */
    {"AmericaToken", new List(){ "USA","Canada" } },

    /* a user with the "AustraliaToken" token will see
    info about Australia */
    {"AustraliaToken", new List(){ "Australia" } },
};
```

Defining the server filter

In the sample project, the server filter is defined in a separate function (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/Controllers/FlexmonsterAPIController.cs#L65-L80). First, a user token is read from the request headers, and then the filter is applied:

```
private ServerFilter GetServerFilter()
{
    // get the user token from the request headers
    HttpContext.Request.Headers.TryGetValue("UserToken", out StringValues userRole);
    if (userRole.Count == 1)
    {
        // create a server filter
        ServerFilter serverFilter = new ServerFilter();
        // specify a field to filter
        serverFilter.Field = new Field() {
            UniqueName = "Country",
            Type = ColumnType.stringType
        };
        // include the members that correspond to the user token
        serverFilter.Include = _userPermissions[userRole[0]];
        return serverFilter;
    }
    return null;
}
```

To learn more about the server filter’s parameters and their types, see the server filter structure (#server-filter-structure).

Applying the server filter

After the filter is specified, it is passed as a parameter to GetMembers() (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/Controllers/FlexmonsterAPIController.cs#L41) and GetAggregatedData() (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/Controllers/FlexmonsterAPIController.cs#L49) methods, for example:

```
public MembersResponse PostMembers([FromBody]MembersRequest request)
{
    var response = _apiService.GetMembers(request, GetServerFilter());
    return new JsonResult(response);
}
```

To apply several server filters, add them to an IEnumerable<ServerFilter> collection and pass it to GetMembers() (https://www.flexmonster.com/doc/controllers-methods-for-request-handling/#!GetMembers) or GetAggregatedData() (https://www.flexmonster.com/doc/controllers-methods-for-request-handling/#!GetAggregatedData) methods.

The server filter structure

The server filter has the following structure:

```
public class ServerFilter
{
    public Field Field { get; set; }
    public List<object> Include { get; set; }
    public List<object> Exclude { get; set; }
}
```

Below is a detailed description of the filter parameters:

- Field – Field. The field to filter. If a non-existent field is specified, Flexmonster.DataServer.Core.dll throws an exception.
- Include – List<object>. Field members to include in the server's response.
- Exclude – List<object>. Field members to exclude from the server's response.

Here is an example of how the server filter can be created:

```
ServerFilter serverFilter = new ServerFilter();
serverFilter.Field = new Field() {
    UniqueName = "Country",
    Type = ColumnType.stringType
};
serverFilter.Include = new List<object>(){ "USA", "Canada" };
```

What's next?

You may be interested in the following articles:

- Implementing the custom parser (/doc/implementing-custom-parser)
- Referencing the Data Server as a DLL (/doc/referencing-data-server-as-dll/)
- Implementing the API controller (/doc/implementing-api-controller/)
- DLL configurations reference (/doc/dll-configurations-reference/)

4.5.6.5. Implementing the custom parser

One of the DLL's main advantages is its ability to visualize data from a custom data source. To use the custom data source with the DLL, you need to implement a special parser.

This guide describes how to create and use a parser for the custom data source:

- Example of a custom parser (#custom-parser-example)
- The IParser interface (#iparser-interface)

Example of a custom parser

To simplify the implementation of the custom parser, we designed a sample ASP.NET Core application (https://github.com/flexmonster/flexmonster-data-server-dll) using Flexmonster.DataServer.Core.dll. This application contains an example of a custom parser, which can be viewed in the CustomParser.cs (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/CustomParser.cs) file.

To create a new ASP.NET Core application with the Data Server or to embed the DLL in an existing project, refer to the guide on Flexmonster.DataServer.Core.dll (/doc/referring-data-server-as-dll/).

Initializing the custom parser

The CustomParser class implements the IParser interface (#iparser-interface).

First, the custom parser's properties are specified:

```
public string Type { get; private set; }

public IndexOptions IndexOptions { get; set; }

public Dictionary<int, ColumnInfo> ColumnInfoByIndex { get; set; }
```

To learn more about these properties, see the IParser interface section (#iparser-interface).

Then, the parser's type is set in the constructor (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/CustomParser.cs#L10) (in this case, the type is "custom"):

```
public CustomParser()
{
    Type = "custom";
}
```

Specifying the data

This parser uses built-in data (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/CustomParser.cs#L46) defined in the data variable. The data has three fields; their names and types are added to the ColumnInfoByIndex collection in the FillDataInfo() method (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/CustomParser.cs#L38):

```
private void FillDataInfo()
{
    ColumnInfoByIndex = new Dictionary<int, ColumnInfo>();
    ColumnInfoByIndex.Add(0, new ColumnInfo("Country", data[0, 0].GetType()));
    ColumnInfoByIndex.Add(1, new ColumnInfo("Price", data[0, 1].GetType()));
    ColumnInfoByIndex.Add(2, new ColumnInfo("Date", data[0, 2].GetType()));
}
```

Parsing the data

The Parse() method is responsible for parsing data. In this case, Parse() (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/CustomParser.cs#L22) calls the FillDataInfo() method first, then processes and returns the data:

```
public IEnumerable<object[,]> Parse()
```

```
{
    FillDataInfo();
    for (int i = 0; i < data.GetLength(0); i++)
    {
        var partData = new object[1, 3];
        for (int j = 0; j < data.GetLength(1); j++)
        {
            partData[0, j] = data[i, j];
        }
        //return data by line
        yield return partData;
    }
}
```

See the IParser interface section (#iparser-interface) to learn more about the Parse() method.

Using the created parser

To be available to the DLL, the custom parser is added to the dependency injection container (https://github.com/flexmonster/flexmonster-data-server-dll/blob/master/DemoDataServerCore/Startup.cs#L29) in the ConfigureServices() method of Startup.cs:

```
services.AddTransient<IParser, CustomParser>();
```

The AddTransient() method is used to provide a new parser instance each time it is requested from the service container.

The IParser interface

All the parsers should implement the IParser interface. IParser has the following definition:

```
public interface IParser
{
    string Type { get; }
    IndexOptions IndexOptions { get; set; }
    IEnumerable<object[,]> Parse();
    Dictionary<int, ColumnInfo> ColumnInfoByIndex { get; }
}
```

Below is a detailed description of the parameters:

- Type – String. The parser’s name. It will be used in the appsettings.json file when creating an index, for example:

  ```
  "DataSources":
  [
      {
  ```

```
        "Type": "custom",
        "Indexes": {
            "custom-index": null
        }
    }
]
```

- Parse – Function. Responsible for parsing the data. The purpose of the function is partial data loading, which allows working with large datasets. The Parse() method has the following signature:

  ```
  IEnumerable<object[,]> Parse();
  ```

  object can have either any primitive type or the DateTime type, so all dates in the dataset to parse should have the DateTime type.
- IndexOptions – IndexOptions. Stores the options defined when creating an index. Index options can be set either via the appsettings.json file or inside the dependency injection container in the ConfigureServices() method of the Startup.cs file:

  ```
  services.Configure<DatasourceOptions>(options =>
  {
      // other configs
      options.Indexes.Add("custom-index", new CustomIndexOptions("custom"));
      // other configs
  });
  ```

  For the custom parser, we recommend setting options in the ConfigureServices() method. This allows adding index-specific options that cannot otherwise be defined in the parser. To specify custom index options, create a class based on the IndexOptions class, for example:

  ```
  public class CustomIndexOptions: IndexOptions
  {
      public string MyProp{ get; set; }

      public JsonIndexOptions(object myProp) : base("custom")
      {
          MyProp= myProp;
      }
  }
  ```

  Then use the class when setting options and in the parser:

  ```
  // in the custom parser
  var customIndexOptions = (CustomIndexOptions)IndexOptions;
  customIndexOptions.MyProp; // to access the custom option
  ```

- ColumnInfoByIndex – Dictionary<int, ColumnInfo>. Allows setting the name and type for columns (note that each column must have the name and type). ColumnInfoByIndex is a collection of [key, value] pairs, where key is the column's number (starting from 0) and value is an instance of the following class:

  ```
  {
  ```

```
    public string Name { get; set; }
    public Type Type { get; set; }
}
```

The ColumnInfoByIndex collection should contain information about columns before the Parse() method returns the data.

What’s next?

You may be interested in the following articles:

- Implementing the server filter (/doc/implementing-server-filter)
- Referencing the Data Server as a DLL (https://www.flexmonster.com/doc/referencing-data-server-as-dll/)
- Implementing the API controller (https://www.flexmonster.com/doc/implementing-api-controller/)
- DLL configurations reference (https://www.flexmonster.com/doc/dll-configurations-reference/)

4.5.6.6. DLL configurations reference

This guide describes the available options for Flexmonster.DataServer.Core.dll. For the Data Server DLL, it is possible to configure data sources and set the data refresh time.

Available configurations

Flexmonster.DataServer.Core.dll can be configured via the ASP.NET configuration file (e.g., appsettings.json). In addition to the Flexmonster Data Server configuration, it can contain any other settings needed for the project, as long as they do not conflict with each other. The Data Server configuration has the following properties:

- "DataSources" – Array of objects. Options for configuring the data sources. Each object has the following properties:
    - "Type" – String. The type of the data source: "json", "csv", "database", or your custom type (for the custom parser (/doc/implementing-custom-parser)).
    - "DatabaseType" (optional) – String. The type of the database: "mysql", "mssql", "postgresql", or "oracle". Only for the "database" data source type.
    - "ConnectionString" (optional) – String. A connection string for the database. Only for the "database" data source type.
    - "Indexes" – Object. Contains a list of datasets. Each dataset is represented by a "key": "value" pair, where "key" is a dataset name, and "value" is an object. The "value" can either be null (only for the custom data source type) or have the following properties:
        - "Path" (optional) – String. The path to the file with data. Only for "json" and "csv" data source types.
        - "Query" (optional) – String. The query to execute (e.g., "SELECT * FROM tablename"). Only for the "database" data source type.
        - "Delimiter" (optional) – String. Defines the specific fields separator to split each CSV row. There is no need to define it if the CSV fields are separated by ,. This property is required only if another character separates fields. *Default value: ","*.
        - "DecimalSeparator" (optional) – String. Defines the specific character used to separate decimal parts of numbers. For example, to import CSV data with commas used to separate decimal parts of numbers (e.g., 3,14), set the "DecimalSeparator" property to ",". *Default value: "."*.
        - "ThousandSeparator" (optional) – String. Defines the specific character used to separate groups of digits in numbers. For example, to import CSV data with periods used to

separate groups of digits in numbers (e.g., 1.000 for one thousand), set the "ThousandSeparator" property to ".". *Default value: ","*.

- "DataStorageOptions" (optional) – Object. Allows configuring options for data storage. It has the following properties:
    - "DataRefreshTime" (optional) – Number. Defines how often the data is reloaded from a file or a database. The refresh time is set in minutes. If "DataRefreshTime" is not specified, the data will not be reloaded.
    - "CacheSizeLimit" (optional) – Number. The maximum number of cached server responses for every index defined in the "DataSources" property. When set to 0, the Data Server does not cache the responses. *Default value: 100*.

Example

Here is an example of a configured appsettings.json file with the custom parser:

```
{
  "DataSources": [
    {
      "Type": "custom-parser",
      "Indexes": {
        "custom-index": null
      }
    },
    {
      "Type": "json",
      "Indexes": {
        "first-json-index": {
          "Path": "data/data.json"
        },
        "second-json-index": {
          "Path": "data/another-data.json"
        }
      }
    },
    {
      "Type": "csv",
      "Indexes": {
        "csv-index": {
          "Path": "data/data.csv",
          "Delimiter": ";",
          "DecimalSeparator": "."
        }
      }
    }
  ],
  "DataStorageOptions": {
    "DataRefreshTime": 60,
    "CacheSizeLimit": 150
  }
}
```

What’s next?

You may be interested in the following articles:

- Referencing the Data Server as a DLL (https://www.flexmonster.com/doc/referencing-data-server-as-dll/)
- Implementing the API controller (https://www.flexmonster.com/doc/implementing-api-controller/)
- Implementing the server filter (https://www.flexmonster.com/doc/implementing-server-filter/)
- Implementing the custom parser (https://www.flexmonster.com/doc/implementing-custom-parser/)
- The controller’s methods for request handling (/doc/controllers-methods-for-request-handling/)

4.5.6.7. The controller's methods for request handling

The API controller’s task is to accept and handle the custom data source API requests (/api/all-requests/) from Flexmonster Pivot. To simplify the request handling, Flexmonster.DataServer.Core.dll provides a set of ready-to-use methods that allow getting fields, members, and aggregated data. These methods belong to the DLL’s IApiService class.

This guide contains the following sections:

- The GetFields() method (#GetFields)
- The GetMembers() method (#GetMembers)
- The GetAggregatedData() method (#GetAggregatedData)

The GetFields() method

The GetFields() method returns a list of all fields with their types (see the response to the /fields request (https://www.flexmonster.com/api/fields-request/#response)). This method has two signatures:

- Schema GetFields(string index)
  Parameters: GetFields with this signature has the same parameter as the /fields request (/api/fields-request/#request).
- Schema GetFields(FieldsRequest request)
  Parameters: the request parameter has the FieldsRequest type. FieldsRequest is a predefined class from the DLL that describes the /fields request’s structure (https://www.flexmonster.com/api/fields-request/#request).

Schema is a class that describes the structure of the response to the /fields request. (/api/fields-request/#response)

The GetMembers() method

The GetMembers() method returns a list of all field members (see the response to the /members request (https://www.flexmonster.com/api/members-request/#response)). This method has the following signatures:

- MembersResponse GetMembers(string index, Field field, int page, LogicFilter filter, [IEnumerable<ServerFilter> serverFilter])
  Parameters: index, field, filter, and page are the same parameters as in the /members request (https://www.flexmonster.com/api/members-request/#request). serverFilter is an optional parameter containing the server filters.
- MembersResponse GetMembers(MembersRequest membersRequest, [IEnumerable<ServerFilter> serverFilter])
  Parameters: membersRequest has the MembersRequest type. MembersRequest is a predefined class from the DLL that describes the /members request’s structure (/api/members-request/#request). serverFilter is an optional parameter containing the server filters.
- MembersResponse GetMembers(MembersRequest membersRequest, ServerFilter serverFilter)

Parameters: membersRequest has the MembersRequest type. MembersRequest is a predefined class from the DLL that describes the /members request's structure (/api/members-request/#request). serverFilter contains the server filters.

MembersResponse is a class that describes the structure of the response to the /members request (/api/members-request/#response).

The GetAggregatedData() method

The GetAggregatedData() method returns the aggregated data (see the response to the /select request (https://www.flexmonster.com/api/select-request-for-pivot-table/#response)). This method has the following signatures:

- SelectResponse GetAggregatedData(string index, Query query, int page, [IEnumerable<ServerFilter> serverFilter])
  Parameters: index, query, and page are the same parameters as in the /select request (https://www.flexmonster.com/api/select-request-for-pivot-table/#request). serverFilter is an optional parameter containing the server filters.
- SelectResponse GetAggregatedData(SelectRequest selectRequest, [IEnumerable<ServerFilter> serverFilter])
  Parameters: selectRequest has the type SelectRequest. SelectRequest is a predefined class from the DLL that describes the /select request's structure (https://www.flexmonster.com/api/select-request-for-pivot-table/#request). serverFilter is an optional parameter containing the server filters.
- SelectResponse GetAggregatedData(SelectRequest selectRequest, ServerFilter serverFilter)
  Parameters: selectRequest has the type SelectRequest. SelectRequest is a predefined class from the DLL that describes the /select request's structure (/api/select-request-for-pivot-table/#request). serverFilter contains the server filter.

SelectResponse is a class that describes the structure of the response to the /select request (/api/select-request-for-pivot-table/#response).

The DLL's methods and classes can be overridden and adjusted to your needs.

What's next?

You may be interested in the following articles:

- Implementing the API controller (https://www.flexmonster.com/doc/implementing-api-controller/)
- Referencing the Data Server as a DLL (https://www.flexmonster.com/doc/referencing-data-server-as-dll/)
- Implementing the server filter (/doc/implementing-server-filter/)
- Implementing the custom parser (/doc/implementing-custom-parser/)
- DLL configurations reference (https://www.flexmonster.com/doc/dll-configurations-reference/)

4.5.7. Troubleshooting the Data Server

This section provides solutions to the errors you might experience while working with Flexmonster Data Server. If you cannot find your error here, post a question to our Help Forum (https://www.flexmonster.com/forum/).

Browser error: 'ERR_HTTP2_INADEQUATE_TRANSPORT_SECURITY'

The main reason for this error is that HTTP/2 does not support the Data Server's SSL certificate. See this thread (https://stackoverflow.com/questions/54865903/visual-studio-2017-and-chrome-err-spdy-inadequate-transport-security-when-start) to learn more.

One of the possible solutions to this error is to set the protocol to HTTP/1.1 in flexmonster-config.json:

```
"HTTPS": {
    "Enabled": true,
    "Protocols": "Http1",
    "Certificate": {
        // your certificate
    }
}
```

Console error: ‘Cannot connect to the database using “Server=xxxx;Port=xxxx;Uid=xxxx;Pwd=xxxx;Database=xxxx”. Please check connection string’

When connecting to a secure PostgreSQL instance with the Data Server, some additional configurations are needed. Learn more details in this forum thread (https://www.flexmonster.com/question/error-while-reading-from-stream/).

Console error: ‘Cannot connect to the database using “Server=(localdb)\\MSSQLLocalDB;Uid=root;Pwd=password;Database=database_name”. Please check connection string. Details: Cannot connect to SQL Server Browser. Ensure SQL Server Browser has been started.’

This error may appear when connecting to an SQL server instance remotely. It means that SQL Server Browser is not started. See how to start SQL Server Browser (https://docs.microsoft.com/en-us/dynamics-nav/how-to--start-sql-browser-service).

For more information on remote connections to SQL Server, refer to our guide (https://www.flexmonster.com/doc/connect-to-mssql-database/#!connect-to-remote-db).

4.6. MongoDB

4.6.1. Introduction to Flexmonster MongoDB Connector

Flexmonster MongoDB Connector is a special server-side tool intended to simplify working with MongoDB. The Connector is easy to set up and use.

To start transferring data from a MongoDB database to Flexmonster, the Connector needs to be embedded into a middle-layer server. This server has to accept requests from Flexmonster and pass them to the Connector.

The Connector is responsible for sending requests to your MongoDB database, fetching data from that database, and passing the data back to the server.

The MongoDB Connector is based on our custom data source API, so all calculations are performed server-side. Since the received data is already aggregated, there is a reduced load on the browser’s memory. The Connector also prepares your data so it fits the Flexmonster Pivot requirements.

For a quick start in using the Connector, see the following guide:

- Getting started with the MongoDB Connector (/doc/how-to-connect-to-mongodb/)

To learn how to embed the Connector into your server, have a look at this tutorial:

- Embedding the MongoDB Connector into the server (/doc/embed-mongodb-connector/)

4.6.2. Getting started with the MongoDB Connector

To quickly and easily connect to a MongoDB database, Flexmonster recommends using our Flexmonster MongoDB Connector that is based on our custom data source API.

The Connector needs to be embedded into a middle-layer server that accepts requests from Flexmonster and passes them to the Connector. The Connector then fetches the required data and sends it back to the server, then finally the data reaches Flexmonster and gets displayed in your browser.

To demonstrate how the Connector, the server, and Flexmonster cooperate, we have prepared a sample ready-to-use project. Using this project as a starting point, you can also connect to your MongoDB database.

Prerequisites

To run this simple application you will need Node.js and npm. Get it here (https://docs.npmjs.com/getting-started/installing-node) if it’s not already installed on your machine.

Run a sample GitHub project

To run our sample project from GitHub, follow these steps:

Step 1. Download the .zip archive with the sample or clone it from GitHub (https://github.com/flexmonster/pivot-mongo) with the following command:

```
git clone https://github.com/flexmonster/pivot-mongo
cd pivot-mongo
```

Step 2. Install the npm packages described in package.json:

```
npm install
```

Step 3. Run the server with the following commands:

```
npm run build
npm run start
```

Now that the server is running and ready to accept Flexmonster’s requests, you can open client/index.html in the browser to see the Connector’s output.

In this tutorial, you connect to Flexmonster's sample database. To connect to your own database, follow the steps from the section below.

Connect to your MongoDB database

You can use the sample project as a starting point to connect to your MongoDB database. To accomplish this, complete the following steps:

Step 1. Open src/controller/mongo.ts and replace the existing connection string with yours:

```
MongoClient.connect("your connection string", {
    ...
});
```

Step 2. Then replace the database name with yours:

```
dbo = db.db("your database name");
```

Step 3. In client/index.html, replace the index value with your collection's name:

```
report: {
    "dataSource": {
        "type": "api",
        "url": "http://localhost:9204/mongo",
        "index": "your-collection-name"
    }
},
```

Step 3. Run the server with the following commands:

```
npm run build
npm run start
```

Now the data from your MongoDB database is displayed when you open the client/index.html file in a browser.

You can take our sample project as a ready template and make it work for your application by adding your logic, improving, and expanding the project.

To learn how to embed the Connector into your server, have a look at this tutorial:

- Embedding the Connector into a server (/doc/embed-mongodb-connector/)

What’s next?

- How to configure which data subset is shown (https://www.flexmonster.com/doc/slice/)
- How to specify functionality available to customers (https://www.flexmonster.com/doc/options/)
- How to define number formatting (https://www.flexmonster.com/doc/number-formatting/)
- How to define conditional formatting (https://www.flexmonster.com/doc/conditional-formatting/)
- How to define a format for date and time (https://www.flexmonster.com/doc/date-and-time-formatting/)

4.6.3. Embedding the MongoDB Connector into the server

In our previous guide (/doc/how-to-connect-to-mongodb/), we launched our sample GitHub project demonstrating the basics of using Flexmonster MongoDB Connector. The information from that guide is considered a prerequisite for this tutorial, it is highly recommended that you complete the previous guide before starting this one.

This tutorial describes how to embed the Connector into a server.

To fetch data from a database and pass it to the component, the Connector needs a mediator between itself and Flexmonster. Your server acts as this mediator and its task is to handle Flexmonster’s requests and to pass them to the Connector in the correct format. The server has to be configured properly to complete this task.

All requests have the type property in the request body. There are 4 types of requests that can be distinguished by the URL path and type value:

- <url>/handshake – The first (handshake) request to establish a connection between the client and server.
- <url>/fields – Request for all fields with their types (i.e., meta-object or schema).
- <url>/members – Request for all members of a field.
- <url>/select – Request for the data.

The value of type will always be the same as the endpoint name, e.g., when a request is sent to <url>/fields, the value of type is "fields".

The Connector has three methods to handle /fields, /members, and /select requests. See the documentation (/api/all-methods/) to learn more about these methods.

The following guide will walk you through the process of server configuration.

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

In the next steps, we will start configuring the server.

Step 2. Create a simple server

Now we will focus on the server part of our project.

First, let’s create a simple server using Express.js (http://expressjs.com/). If you already have a server, jump to

Step 3. Create a separate module for the request routing (#step-3).

Follow the steps below to get a simple server application:

Step 2.1. Create a folder for the server and run the npm init command there to generate the package.json file.

Step 2.2. The server needs the express, cors, and body-parser packages. Install them with npm:

Install Express.js

```
npm install express
```

Install CORS

```
npm install cors
```

Install body parser

```
npm install body-parser
```

Step 2.3. Then, create a simple server in a .js file (e.g., server.js):

```
const express = require("express");
var cors = require('cors');
var bodyParser = require('body-parser');

const app = express();

app.use(cors());
app.use(bodyParser.json());

var port = 9204;
app.listen(port, () => {
    console.log(Example app listening at http://localhost:${port})
});
```

Step 3. Create a separate module for the request routing

All the request routing will be implemented in a separate .js file (e.g., mongo.js).

At this step, include the .js file (i.e., mongo.js) into your server. All the requests coming to <url> will be handled by this separate module (in this case, the <url>/mongo path is used):

```
app.use('/mongo', require('./mongo.js'));
```

Now, all the requests sent to <url> (in this case,<url>/mongo) will be processed by the mongo module.

However, since we haven't yet defined the mongo.js file properly, the code won't work just yet.

If Flexmonster Pivot Table is running on a different server, enable CORS.

Step 4. Install packages for the mongo.js module

Install Flexmonster MongoDB Connector and the MongoDB driver:

```
npm install flexmonster-mongo-connector
npm install mongodb
```

Now it's time to configure the mongo.js module.

Step 5. Configure the mongo.js module

Configure the mongo.js module in three steps:

Step 5.1. In the JavaScript file mentioned in the previous step (i.e., mongo.js), include the following libraries:

```
const mongo = require('express').Router();
const mongodb = require("mongodb");
const fm_mongo_connector = require("flexmonster-mongo-connector");
```

Step 5.2. Then set up a connection with your MongoDB database and define the Connector:

```
let dbo = null;
let _apiReference = null; // it'll be the Connector instance

mongodb.MongoClient.connect("mongodb://read:only@olap.flexmonster.com:27017", {
    useNewUrlParser: true
    }, (err, db) => {
        if (err)
            throw err;
        dbo = db.db("flexmonster");
        _apiReference = new fm_mongo_connector.MongoDataAPI();
});
```

Step 5.3. Export the mongo module so the server can use it:

```
// requests handling functions will appear here
module.exports = mongo;
```

This module can now connect to your MongoDB database, but it does not handle the Flexmonster Pivot requests. In the next steps, we will handle all Flexmonster requests.

Step 6. (optional) Handle the /handshake request

The /handshake request establishes a connection between the client and server sides. If the server sends the implemented version of the custom data source API in response to the /handshake request, the client can check whether the server and the client implement the same version of the custom data source API.

To receive notifications about version compatibility, respond to the /handshake request with the implemented version of the custom data source API:

```
mongo.post("/handshake", async (req, res) => {
    try {
        res.json({ version: _apiReference.API_VERSION });
    } catch (err) {
         handleError(err, res);
    }
});
```

Handing the /handshake (https://www.flexmonster.com/api/handshake-request/) request is optional. Flexmonster Pivot will proceed to the next request even if the server does not handle it.

Step 7. Handle the /fields request

The next request that needs to be handled is a /fields (https://www.flexmonster.com/api/fields-request/) request; it is sent to <url>/fields. It can be handled like this:

```
mongo.post("/fields", async (req, res) => {
    try {
        const result = await _apiReference.getSchema(dbo, req.body.index);
        res.json(result);
    } catch (err) {
        //your error handler
    }
});
```

When Flexmonster Pivot successfully receives a response to the /fields request, it shows the Field List with all of the available fields.

To learn more about the getSchema method, see our documentation (https://www.flexmonster.com/api/getschema/).

Step 8. Handle the /members request

The next request to handle is the request for the field’s members (https://www.flexmonster.com/api/members-request/) that is sent to <url>/members. It can be handled like this:

```
mongo.post("/members", async (req, res) => {
    try {
        const result = await _apiReference.getMembers(
            dbo, req.body.index, req.body.field,
            { page: req.body.page, pageToken: req.body.pageToken });
        res.json(result);
    } catch (err) {
      //your error handler
    }
});
```

When Flexmonster Pivot successfully receives the response to this request, you will be able to select a string field for rows or for columns and retrieve its members.

To learn more about the getMembers method, see our documentation (https://www.flexmonster.com/api/getmembers-mongo/).

Step 9. Handle the /select request

When a field is selected for rows and/or columns and a numeric field is selected for measures in the Field List, the /select (https://www.flexmonster.com/api/select-request-for-pivot-table/) request is sent to the endpoint <url>/select. It can be handled like this:

```
mongo.post("/select", async (req, res) => {
    try {
        const result = await _apiReference.getSelectResult(
            dbo, req.body.index, req.body.query,
            { page: req.body.page, pageToken: req.body.pageToken });
        res.json(result);
    } catch (err) {
      //your error handler
    }
});
```

When Flexmonster successfully receives the response to this kind of /select request, a pivot table with the received data is shown.

To learn more about the getSelectResult method, see our documentation (https://www.flexmonster.com/api/getselectresult/).

Step 10. Run the server

Run your server with the following command:

```
node server.js
```

Step 11. Configure the report

Now it's time to configure the pivot table on the web page. In report.dataSource, define these parameters to connect to your server:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "api",
            url: "<url>",
            index: "your-collection-name"
        }
    }
});
```

Here, url is the path to your API endpoints (e.g., "http://localhost:9204/mongo"), and index is your collection's name. index will be sent with every request.

Open your HTML page in the browser to see the result – the pivot table with data from your MongoDB database is shown.

What's next?

- How to configure which data subset is shown (https://www.flexmonster.com/doc/slice/)
- How to specify functionality available to customers (https://www.flexmonster.com/doc/options/)
- How to define number formatting (https://www.flexmonster.com/doc/number-formatting/)
- How to define conditional formatting (https://www.flexmonster.com/doc/conditional-formatting/)
- How to define a format for date and time (https://www.flexmonster.com/doc/date-and-time-formatting/)

4.7. Microsoft Analysis Services

4.7.1. Connecting to Microsoft Analysis Services

Flexmonster supports both tabular and multidimensional model types. There are two ways to connect to Microsoft Analysis Services using Flexmonster Pivot:

1. Via XMLA (/doc/connecting-to-microsoft-analysis-services/#xmla) – an industry standard for data access in analytical systems. Works for multidimensional models only.
2. Via Flexmonster Accelerator (/doc/getting-started-with-accelerator-ssas/) – a special server-side utility developed by Flexmonster. Works for both multidimensional and tabular models.

If you already have XMLA configured (msmdpump.dll) it will be easier to start with option #1. Option #2 is the better choice if you:

- Do not have XMLA.
- Use a tabular model.
- Need some advanced features (such as increased loading speeds, use of credentials, etc.).
- Use Azure Analysis Services.

Connecting to Microsoft Analysis Services via XMLA

XMLA (XML for Analysis) is an industry standard for data access in analytical systems such as OLAP and Data Mining.

Follow the steps below to configure a connection to Microsoft Analysis Services via XMLA. To work with tabular models or Azure Analysis Services, use Flexmonster Accelerator (https://www.flexmonster.com/doc/getting-started-with-accelerator-ssas/).

Step 1: Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2: Configure XMLA access to the cube

Skip this step if you already have XMLA configured. Otherwise refer to this article: how to set up an HTTP endpoint for accessing an Analysis Services instance (https://msdn.microsoft.com/en-us/library/gg492140.aspx).

Step 3: Enable cross-origin resource sharing (CORS)

By default, the browser prevents JavaScript from making requests across domain boundaries. CORS allows web applications to make cross-domain requests. Follow these step-by-step instructions to enable CORS for IIS (/question/stream-error-occurred-while-loading-httplocalhost8080olapmsmdpump-dll) to be able to read data from Microsoft Analysis Services.

Step 4: Configure the report with your own data

Now it’s time to configure the pivot table on the web page. Let’s create a minimal report for this (replace proxyUrl, catalog, and cube parameters with your specific values):

```
var pivot = new Flexmonster({
    container: "pivotContainer", \
    toolbar: true,
    report: {
        dataSource: {
            type: "microsoft analysis services",
            /* URL to msmdpump.dll */
            proxyUrl: "https://olap.flexmonster.com/olap/msmdpump.dll",
            /* Catalog name */
            catalog: "Adventure Works DW Standard Edition",
            /* Cube name */
            cube: "Adventure Works",
        }
    }
});
```

Launch the web page from a browser — there you go! A pivot table is embedded in your project. Check out an example on JSFiddle (https://jsfiddle.net/flexmonster/3z3abpfs/).

Optimize data loading

The subquery parameter helps Flexmonster take less time for loading and rendering. Below is an example for showing reports for a specific year from the date hierarchy:

```
{
    "dataSource": {
        "type": "microsoft analysis services",
        "proxyUrl": "https://olap.flexmonster.com/olap/msmdpump.dll",
        "catalog": "Adventure Works DW Standard Edition",
        "cube": "Adventure Works",
        "subquery": "select {
                     [Delivery Date].[Calendar].[Calendar Year].&[2011]
                     } on columns from [Adventure Works]"
    },
    "slice": {
        "rows": [ { "uniqueName": "[Delivery Date].[Calendar]" } ],
        "columns": [
            { "uniqueName": "[Product].[Category]" },
            { "uniqueName": "[Measures]" }
        ],
        "measures": [ { "uniqueName": "[Measures].[Order Count]" } ]
    }
}
```

Try it on JSFiddle (https://jsfiddle.net/flexmonster/3dmbzyp5/).

What's next?

You may be interested in the following articles:

- Configuring the authentication process (https://www.flexmonster.com/doc/configuring-authentication-process/)
- Getting started with Flexmonster Accelerator (https://www.flexmonster.com/doc/getting-started-with-accelerator-ssas/)

4.7.2. Getting started with Flexmonster Accelerator

Flexmonster Accelerator for Microsoft Analysis Services cubes is a special server-side proxy that increases data loading speeds from the server to the customer's browser. Flexmonster Accelerator works with:

- Microsoft Analysis Services multidimensional cubes.
- Microsoft Analysis Services tabular models.
- Azure Analysis Services cubes.

When working with OLAP cubes, a browser component communicates with the server via the XMLA protocol. The XMLA protocol is heavy and exchanges a lot of excessive information – it takes too much time and memory to load and process the data.

We replaced the XMLA protocol and use direct requests from the component to the server.

This is our solution to two major problems that many of those who work with big data face:

- We made big data transfer from the server to the browser incredibly fast. Our tool allows you to transfer large multidimensional data super easily and quickly. Reporting becomes more enjoyable and prompt for your end users.
- We greatly reduced the load on the browser memory.

Open Live Demo (https://www.flexmonster.com/demos/connect-msas)

There are two ways to use Flexmonster Accelerator:

- Install the Accelerator as a Windows Service (#use-as-windows-service)
- Reference the Accelerator as a DLL (https://www.flexmonster.com/doc/referencing-accelerator-as-a-dll/)

Install Flexmonster Accelerator as a Windows Service

The main benefits of running the Accelerator as a Windows service are:

- It runs in the background and out of sight.
- It starts automatically on Windows startup.
- It's hard for a user to inadvertently quit the application.

Prerequisites

- Flexmonster Pivot version 2.2 or higher
- Microsoft Analysis Services installed and configured
- Microsoft .NET Framework 4.5.2 or higher
- Flexmonster CLI

Flexmonster CLI (/doc/cli-overview/) is a command-line interface tool for Flexmonster. Install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

After that, a new flexmonster command will be available in the console. Learn more about Flexmonster CLI and its commands in our documentation (https://www.flexmonster.com/doc/cli-overview/).

Now follow the steps below to start using Flexmonster Accelerator as a Windows service.

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Install Flexmonster Accelerator

Start the installation process with the following CLI command:

```
flexmonster add accelerator -i
```

The flexmonster add accelerator command will do the following:

- Download the .zip archive with Flexmonster Accelerator.
- Automatically unpack the files in the current folder — as a result, the flexmonster-accelerator/ folder will appear in your current directory.

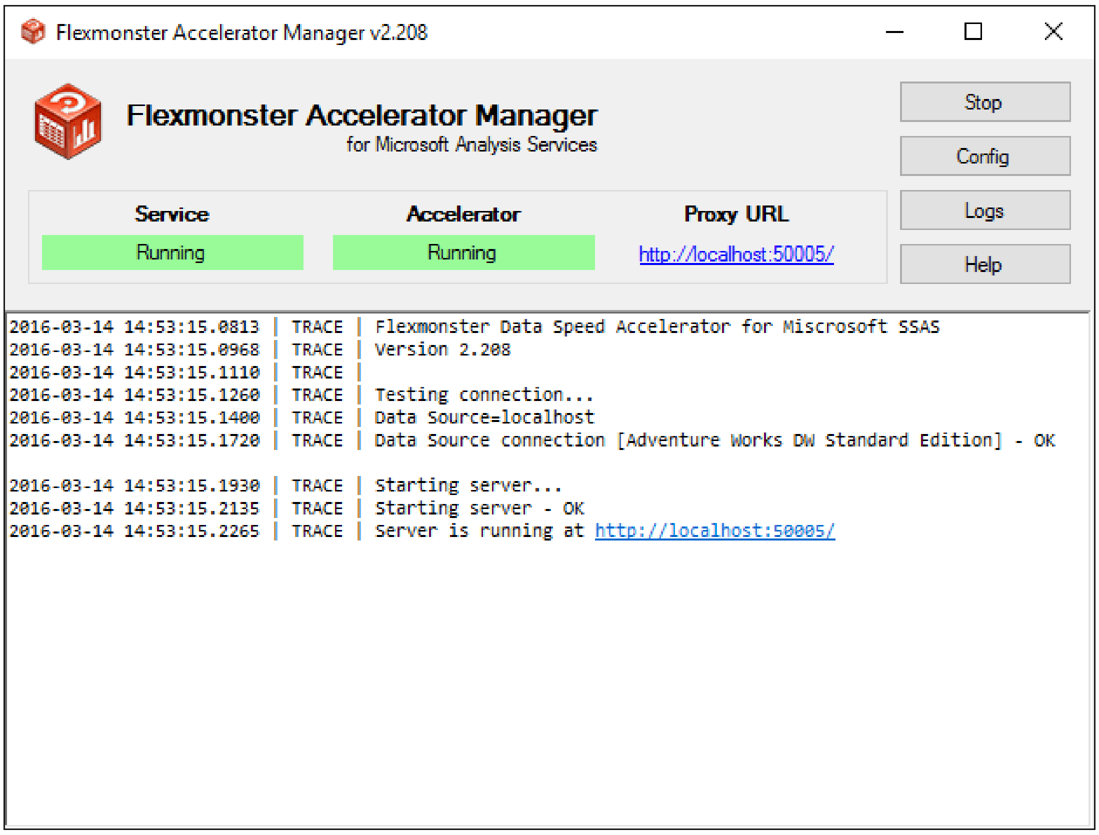
The -i option, which stands for --install, will automatically initiate the installation using the Flexmonster Accelerator.msi setup file. When the installation begins, just follow the wizard.

The flexmonster-accelerator/ folder has the following structure:

- Flexmonster Accelerator.msi – a server-side utility that handles the connection between Microsoft Analysis Services and Flexmonster Pivot.
- flexmonster.config – a file that contains configuration parameters for the utility (connection string, port, etc.).

Step 3. Configure Flexmonster Accelerator on the server

After a successful installation, run **Flexmonster Accelerator Manager**:


Flexmonster Accelerator Manager v2.208
Flexmonster Accelerator Manager
for Microsoft Analysis Services
Stop
Config
Logs
Help
Service
Running
Accelerator
Running
Proxy URL
http://localhost:50005/

```
2016-03-14 14:53:15.0813 | TRACE | Flexmonster Data Speed Accelerator for Miscrosoft SSAS
2016-03-14 14:53:15.0968 | TRACE | Version 2.208
2016-03-14 14:53:15.1110 | TRACE |
2016-03-14 14:53:15.1260 | TRACE | Testing connection...
2016-03-14 14:53:15.1400 | TRACE | Data Source=localhost
2016-03-14 14:53:15.1720 | TRACE | Data Source connection [Adventure Works DW Standard Edition] - OK

2016-03-14 14:53:15.1930 | TRACE | Starting server...
2016-03-14 14:53:15.2135 | TRACE | Starting server - OK
2016-03-14 14:53:15.2265 | TRACE | Server is running at http://localhost:50005/
```

Then, configure the Accelerator using the flexmonster.config file. It contains the following parameters:

- CONNECTION_STRING – (**required**) the connection string for Microsoft Analysis Services. Example: Data Source=localhost;.
- PORT – (optional) the port number for the proxy service endpoint. *Default value: 50005*.
- CACHE_MEMORY_LIMIT – (optional) the maximum memory size available for caching (in MB). *Default value: 0 (unlimited)*.
- CACHE_ENABLED – (optional) indicates whether caching is enabled. Available since version 2.211. *Default value: true*.
- GZIP – (optional) indicates whether server response compression is enabled. Available since version 2.409. *Default value: true*.
- HTTPS – (optional) indicates whether the HTTPS connection is enabled. *Default value: false*.
- WINDOWS_AUTH – (optional) indicates whether Windows authentication is enabled. *Default value: false*.
- ALLOW_ORIGIN – (optional) the origin from which the server accepts the requests. If ALLOW_ORIGIN is set to *, requests from all origins are accepted. Note that if authentication is enabled (WINDOWS_AUTH=true), * cannot be set as the origin. In this case, specific origins must be defined. Several origins can be defined as follows: ALLOW_ORIGIN = http://localhost, https://example.com. *Default value: *.*

Note: to connect to Azure Analysis Services with Flexmonster Accelerator, specify the following parameters in flexmonster.config:

- AZURE_AUTHORITY – the Azure authority URL (e.g., https://login.microsoftonline.com/<your_domain>). Learn more in the Microsoft guide (https://docs.microsoft.com/en-us/rest/api/datacatalog/authenticate-a-client-app#what-you-need-to-authenticate-a-data-catalog-client-app).
- AZURE_CLIENT_ID – the client ID (https://docs.microsoft.com/en-us/rest/api/datacatalog/register-a-client-app#get-the-application-client-id) of the service principal registered in Azure Active Directory.
- AZURE_CLIENT_SECRET – the client secret (https://docs.microsoft.com/en-us/azure/active-directory/develop/howto-create-service-principal-portal#option-2-create-a-new-application-secret) of the

service principal registered in Azure Active Directory.

You can check whether the Accelerator is up and running by navigating to its URL in the browser (by default: http://localhost:50005).

After the configuration you can close **Flexmonster Accelerator Manager** – the service will continue working in the background.

Step 4. Open a port for the Accelerator in the firewall

If you plan to allow connections to the Accelerator from outside the server, open the appropriate port in the firewall. The default port number is 50005, but it can be changed using the PORT parameter in the flexmonster.config file.

Step 5. Configure the component

Now it's time to configure the client – Flexmonster Pivot Table and Charts. Let's create a minimal configuration using the JavaScript API (replace proxyUrl, catalog and cube parameters with your specific values):

Connect to a multidimensional model

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "microsoft analysis services",

            /* URL to Flexmonster Accelerator */
            proxyUrl: "http://localhost:50005",

            /* Catalog name */
            catalog: "Adventure Works DW Standard Edition",

            /* Cube name */
            cube: "Adventure Works",

            // Flag to use the Accelerator instead of XMLA protocol binary
            binary: true
        }
    }
});
```

Connect to a tabular model

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "microsoft analysis services",
            /* URL to Flexmonster Accelerator */
            proxyUrl: "http://localhost:50005",

            /* Database name */
            catalog: "Adventure Works DW Standard Edition",

            /* Model name */
            cube: "Adventure Works",

            // Flag to use the Accelerator instead of XMLA protocol binary
            binary: true
        }
    }
});
```

See the full code on JSFiddle (https://jsfiddle.net/flexmonster/xhrwqbkm/).

Now launch the web page from a browser — there you go! A pivot table is embedded in your project.

Cache control

Usually, a cache can improve performance greatly. However, if the underlying database ever changes, the cache goes out of date. By default, caching is enabled and controlled by the Accelerator.

It is also possible to disable the cache with the CACHE_ENABLED parameter in the flexmonster.config file (available since version 2.211):

```
CACHE_ENABLED = false
```

What’s next?

You may be interested in the following articles:

- Configuring the authentication process (/doc/configuring-authentication-process/)
- Configuring a secure HTTPS connection (/doc/configuring-secure-https-connection/)

4.7.3. Referencing the Accelerator as a DLL

Overview

Flexmonster Accelerator can be integrated into your website back end as a separate ASP.NET controller. The main benefits of referencing the Accelerator as a DLL directly from the ASP.NET project are:

- No running of Accelerator installers.
- Easy updates to the newest version.
- No firewall settings.
- No service dependencies.

Prerequisites

- Flexmonster Pivot version 2.4 or higher
- Microsoft Analysis Services installed and configured
- Microsoft .NET Framework 4.5.2 or higher
- Visual Studio IDE

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Configure Flexmonster Accelerator on the server

Choose one of the following options:

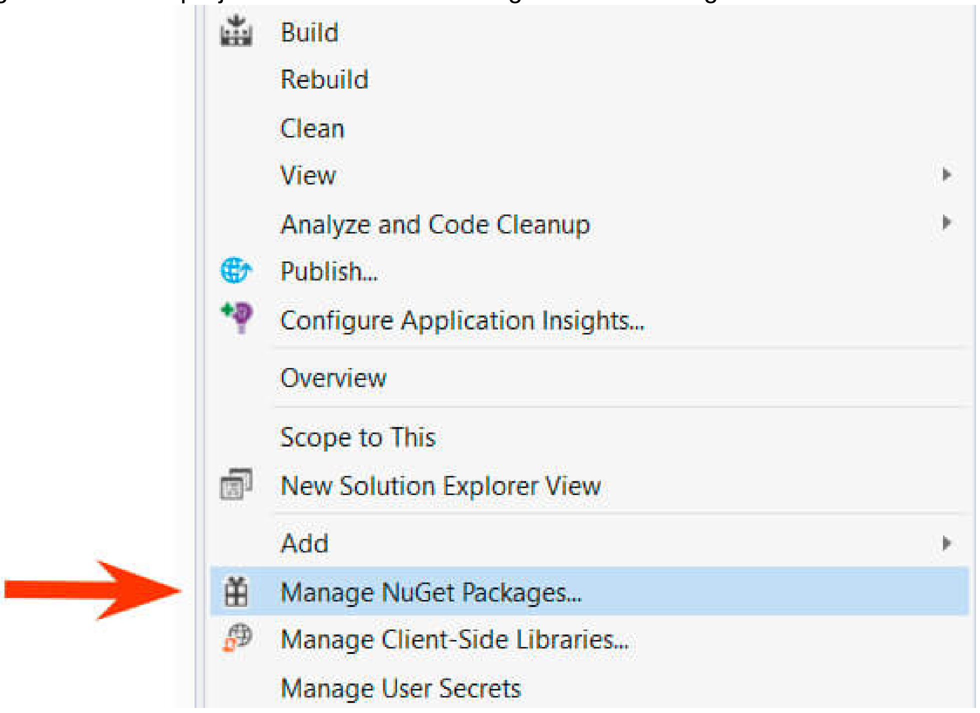
- Use an example GitHub project (https://github.com/flexmonster/pivot-accelerator-dll/)
- Integrate into an existing project (#integrate)

Integrate Flexmonster Accelerator into an existing project

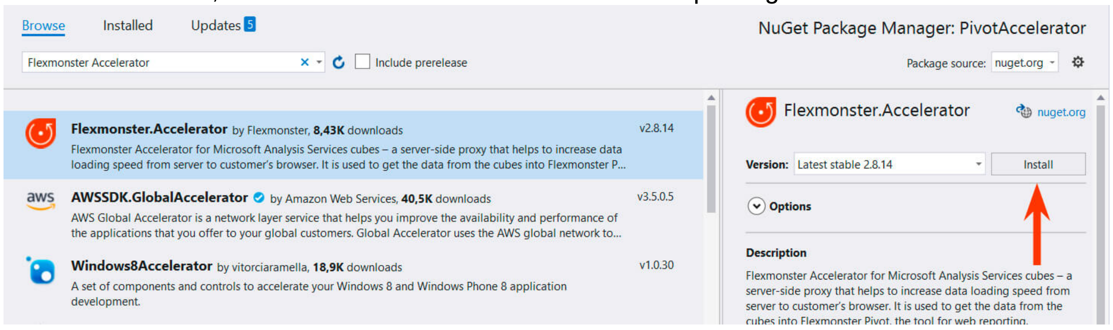
To integrate Flexmonster Accelerator into an existing project, follow these steps:

Step 1. Install the Accelerator with NuGet (https://www.nuget.org/) – the Visual Studio’s package manager:

- Right-click on the project and select the Manage NuGet Packages item:



- In the Browse tab, search for the Flexmonster.Accelerator package and install it:



Browse Installed Updates 5 NuGet Package Manager: PivotAccelerator

Flexmonster Accelerator Include prerelease Package source: nuget.org

Flexmonster.Accelerator by Flexmonster, **8,43K** downloads v2.8.14
Flexmonster Accelerator for Microsoft Analysis Services cubes – a server-side proxy that helps to increase data loading speed from server to customer's browser. It is used to get the data from the cubes into Flexmonster P...

AWSSDK.GlobalAccelerator by Amazon Web Services, **40,5K** downloads v3.5.0.5
AWS Global Accelerator is a network layer service that helps you improve the availability and performance of the applications that you offer to your global customers. Global Accelerator uses the AWS global network to...

Windows8Accelerator by vitorciaramella, **18,9K** downloads v1.0.30
A set of components and controls to accelerate your Windows 8 and Windows Phone 8 application development.

Flexmonster.Accelerator nuget.org

Version: Latest stable 2.8.14 Install

Options

Description
Flexmonster Accelerator for Microsoft Analysis Services cubes – a server-side proxy that helps to increase data loading speed from server to customer's browser. It is used to get the data from the cubes into Flexmonster Pivot, the tool for web reporting.

Step 2. Create a FlexmonsterConfig class with information about the connection:

- ConnectionString – a connection string for Microsoft Analysis Services. Example: Data Source=localhost;.
- CacheManager.MemoryLimit – the maximum memory size available for caching (in bytes).
- CacheManager.Enabled – indicates whether the cache is enabled.

Note: to connect to Azure Analysis Services, specify AuthHelper in FlexmonsterConfig. This property is an instance of the AzureAuthHelper class that has the following parameters:

- ResourceURL – the URL of your Azure Analysis Services instance.
- AzureAuthority – the Azure authority URL (e.g., https://login.microsoftonline.com/<your_domain>). Learn more in the Microsoft guide (https://docs.microsoft.com/en-us/rest/api/datacatalog/authenticate-a-client-app#what-you-need-to-authenticate-a-data-catalog-client-app).
- AzureClientId – the client ID (https://docs.microsoft.com/en-us/rest/api/datacatalog/register-a-client-app#get-the-application-client-id) of the service principal registered in Azure Active Directory.
- AzureClientSecret – the client secret (https://docs.microsoft.com/en-us/azure/active-directory/develop/howto-create-service-principal-portal#option-2-create-a-new-application-secret) of the service principal registered in Azure Active Directory.

Your code should look similar to the following:

```
public class FlexmonsterConfig {
    public static void Register() {
        // Replace with actual data source
        // Example: Data Source=localhost
        Flexmonster.Accelerator.Controllers.FlexmonsterProxyController
        .ConnectionString = "Data Source=localhost";

        Flexmonster.Accelerator.Utils.CacheManager
        .MemoryLimit = 10 * 1024 * 1024; // MB to bytes

        Flexmonster.Accelerator.Utils.CacheManager.Enabled = true;


        /* Specify the AuthHelper property if connecting
           to Azure Analysis Services */
        Flexmonster.Accelerator.Controllers.FlexmonsterProxyController
        .AuthHelper = new AzureAuthHelper(
            // ResourceURL
            "https://westeurope.asazure.windows.net",
            // AzureAuthority
            "https://login.microsoftonline.com/<your_azure_domain>",
            // AzureClientId
            "00000000-0000-0000-0000-000000000000",
            // AzureClientSecret
            "XXXXXXXXXXXXX"
        );
    }
}
```

In our project, this file is located in the Flexmonster Accelerator MVC/App_Start/ folder.

Step 3. Register FlexmonsterConfig inside the Application_Start method of the Global.asax.cs file:

```
FlexmonsterConfig.Register();
```

Step 4. Create an AcceleratorController class that extends FlexmonsterProxyController. This class will handle the requests to the Accelerator:

```
public class AcceleratorController :
    Flexmonster.Accelerator.Controllers.FlexmonsterProxyController {
    public override void OnRequest(BaseArgs args) {
        base.OnRequest(args);
    }
}
```

In our project, this file is located in the Flexmonster Accelerator MVC/Controllers/ folder.

Step 3. Configure the component

Now it’s time to configure the client – Flexmonster Pivot Table and Charts. Let’s create a minimal configuration using the JavaScript API (replace proxyUrl, catalog and cube parameters with your specific values):

Connect to a multidimensional model

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "microsoft analysis services",

            /* URL to Flexmonster Accelerator */
            proxyUrl: "http://localhost:50005",

            /* Catalog name */
            catalog: "Adventure Works DW Standard Edition",

            /* Cube name */
            cube: "Adventure Works",
            // Flag to use the Accelerator instead of XMLA protocol binary
            binary: true
        }
    }
});
```

Connect to a tabular model

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "microsoft analysis services",

            /* URL to Flexmonster Accelerator */
            proxyUrl: "http://localhost:50005",

            /* Database name */
            catalog: "Adventure Works DW Standard Edition",

            /* Model name */
            cube: "Adventure Works",

            // Flag to use the Accelerator instead of XMLA protocol binary
            binary: true
        }
    }
});
```

Launch the web page from a browser — there you go! A pivot table is embedded into your project.

What’s next?

You may be interested in the following articles:

- Configuring the authentication process (https://www.flexmonster.com/doc/configuring-authentication-process/)
- Configuring a secure HTTPS connection (https://www.flexmonster.com/doc/configuring-secure-https-connection/)

4.7.4. Configuring the authentication process

This tutorial explains how to manage the authentication process when working with SQL Server Analysis Services (SSAS).

We support three different approaches:

1. Using roles from Analysis Services (#roles) – this approach is the easiest and it works for both XMLA and the Accelerator.
2. Using Windows authentication:
   - For the XMLA connection (#windows-auth-xmla)
   - For Flexmonster Accelerator (#windows-auth-accelerator)
3. Using custom authorization (#custom-authorization) – works for the Accelerator, recommended for those who already have an ASP.NET portal that handles authorization.

1. Using roles from Analysis Services

In SQL Server Analysis Services, access rights are provided based on roles. More information about role configuration can be found in this tutorial from Microsoft (https://docs.microsoft.com/en-us/sql/analysis-services/multidimensional-models/roles-and-permissions-analysis-services).

After roles are configured in Analysis Services, they can be specified in Flexmonster reports by using the roles property. This property is available for both XMLA and the Accelerator. The following sample demonstrates how to specify roles:

```
{
    dataSource: {
        type: "microsoft analysis services",
        /* URL to msmdpump.dll */
        proxyUrl: "https://olap.flexmonster.com/olap/msmdpump.dll",
        catalog: "Adventure Works DW Standard Edition",
        cube: "Adventure Works",
        /* roles from SSAS, you can add multiple
           roles separated by comma */
        roles: "Sales Manager US"
    }
}
```

Open the example on JSFiddle (https://jsfiddle.net/flexmonster/7g82cnn3/).

2.1. Using Windows authentication for XMLA

Starting from version 2.8.2, Windows authentication is available when connecting to SSAS via XMLA.

Follow the steps below to configure Windows authentication for the XMLA connection.

Step 1. Configure XMLA access to the cube

Skip this step if you already have XMLA configured. Otherwise refer to this article: how to set up an HTTP endpoint for accessing an Analysis Services instance (https://msdn.microsoft.com/en-us/library/gg492140.aspx).

Step 2. Allow the OPTIONS request

In CORS, the browser sends the OPTIONS preflight request to the server. This request determines which request methods and headers the server allows.

The OPTIONS preflight request cannot contain any credentials, so Windows Integrated Authentication will reject this request and ask for authentication. Thus, the server should always accept the preflight request. To allow the

OPTIONS request, see the following guide: CORS tutorial (https://docs.microsoft.com/uk-ua/archive/blogs/friis/putting-it-all-together-cors-tutorial).

Step 3. Configure the data source on the client

Windows authentication should be allowed on the client side as well. To enable the authentication on the client, set the withCredentials property of the Data Source Object (/api/data-source-object/) to true:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "microsoft analysis services",
            proxyUrl: "https://localhost/OLAP/msmdpump.dll",
            catalog: "Adventure Works DW Standard Edition",
            cube: "Adventure Works",
            withCredentials: true
        }
    }
});
```

After applying the configurations, the requests to Microsoft Analysis Services will be performed using your current Windows user.

2.2. Using Windows authentication for Flexmonster Accelerator

Starting from version 2.8.5, Windows authentication is supported for Flexmonster Accelerator.

Follow the guides below to configure Windows authentication for your type of the Accelerator.

Accelerator as a Windows Service

Step 1. Enable authentication on the server

Windows authentication on the server side can be enabled in the flexmonster.config file – a special configuration file for Flexmonster Accelerator. In this file, set the WINDOWS_AUTH property to true to enable the authentication:

```
WINDOWS_AUTH=true
```

Step 2. Specify origins from which requests should be accepted

When enabled, the authentication requires certain origins to be defined in the Access-Control-Allow-Origin header. Origin is a domain that sends requests to Flexmonster Accelerator (e.g., http://localhost:8080 or https://example.com). To allow the origin to send requests to the Accelerator, specify the ALLOW_ORIGIN property in the flexmonster.config file:

```
ALLOW_ORIGIN=http://localhost:8080
```

Several origins can be defined as follows:

```
ALLOW_ORIGIN=http://localhost:8080, https://example.com
```

Step 3. Configure the data source on the client

Windows authentication should be allowed on the client side as well. To enable the authentication on the client, set the withCredentials property of the Data Source Object (https://www.flexmonster.com/api/data-source-object/) to true:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "microsoft analysis services",
            proxyUrl: "http://localhost:50005",
            catalog: "Adventure Works DW Standard Edition",
            cube: "Adventure Works",
            binary: true,
            withCredentials: true
         }
    }
});
```

To apply the configurations, restart Flexmonster Accelerator. You can check if the Accelerator is up and running by navigating to its URL in the browser (http://localhost:50005 (http://localhost:50005/) by default).

Flexmonster Accelerator will automatically use your current Windows user to perform impersonated requests to Microsoft Analysis Services.

If the page with Flexmonster Pivot is opened in the Incognito browser window, the pop-up window prompting to enter your login and password should appear. After you log in with your Windows user credentials, Flexmonster Accelerator should successfully connect to the data source.

Accelerator as a DLL

This guide contains the following sections:

- Run the sample project with Windows authentication (#github-sample)
- Configure Windows authentication in your application (#configure-auth)

Run the sample project with Windows authentication

To illustrate how Windows authentication can be configured in an Accelerator DLL project, we added the windows-authentication branch to our sample GitHub application (https://github.com/flexmonster/pivot-accelerator-dll).

To run the sample project, complete these steps:

Step 1. Download the .zip archive with the windows-authentication branch or clone it from GitHub with the following command:

```
git clone -b windows-authentication https://github.com/flexmonster/pivot-accelerator-dll
```

Step 2. Open the sample project in Visual Studio using the Flexmonster Accelerator MVC.sln solution file.

Step 3. Update the NuGet packages if needed:

1. Right-click the project name in Solution Explorer and select Manage NuGet Packages in the context menu.
2. Go to the Updates tab and check the Select all packages checkbox.
3. Click the Update button.

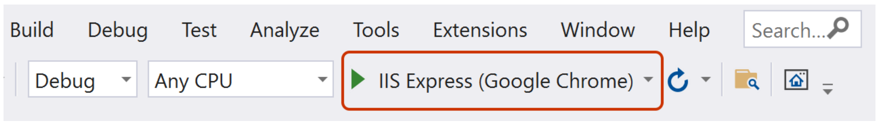
Step 4. Open the FlexmonsterConfig.cs file (https://github.com/flexmonster/pivot-accelerator-dll/blob/windows-authentication/Flexmonster%20Accelerator%20MVC/App_Start/FlexmonsterConfig.cs#L15) and specify your data source (e.g., localhost):

```
public static void Register()
{
  Flexmonster.Accelerator.Controllers.FlexmonsterProxyController
  .ConnectionString = "Data Source=yourDataSource";
  // Other configurations
}
```

Step 5. Now go to the Index.cshtml view (https://github.com/flexmonster/pivot-accelerator-dll/blob/windows-authentication/Flexmonster%20Accelerator%20MVC/Views/Home/Index.cshtml#L25-L26) and set your values for the catalog and cube properties:

```
dataSource: {
    dataSourceType: "microsoft analysis services",
    proxyUrl: "/api/accelerator/",
    catalog: "Your Catalog",
    cube: "Your Cube",
    binary: true,
    withCredentials: true
}
```

Step 6. Run the sample project by clicking the IIS Express button on the toolbar:



To see the result, open http://localhost:55158/ in your browser. Flexmonster Accelerator will automatically use your current Windows user to perform impersonated requests to Microsoft Analysis Services.

Configure Windows authentication in your application

Now let’s see which configurations you should set to enable Windows authentication in your project:

1. Enable authentication on the Accelerator’s side in the FlexmonsterConfig.cs file (https://github.com/flexmonster/pivot-accelerator-dll/blob/windows-authentication/Flexmonster%20Accelerator%20MVC/App_Start/FlexmonsterConfig.cs#L16-L17):

   ```
   public static void Register()
   {
     // Other configurations
     Flexmonster.Accelerator.Controllers.FlexmonsterProxyController
     .AuthEnabled = true;
     Flexmonster.Accelerator.Controllers.FlexmonsterProxyController
     .Impersonator = new WindowsImpersonatorFactory();
     // Other configurations
   }
   ```

2. In the Web.config file (https://github.com/flexmonster/pivot-accelerator-dll/blob/windows-authentication/Flexmonster%20Accelerator%20MVC/Web.config#L93-L104), specify the <security> element to define authentication configurations for the server:

   ```
   <system.webServer>
     <!-- Other tags -->
     <security>
       <authentication>
         <anonymousAuthentication enabled="true" />
         <windowsAuthentication enabled="true" />
       </authentication>
       <authorization>
         <remove users="*" roles="" verbs=""/>
         <add accessType="Allow" users="?" verbs="OPTIONS"/>
         <add accessType="Deny"
          users="?" verbs="GET,PUT,POST,DELETE"
         />
         <add accessType="Allow" roles="Users" />
       </authorization>
     </security>
     <!-- Other tags -->
   </system.webServer>
   ```

   Learn more about the <security> element in the Microsoft documentation (https://docs.microsoft.com/en-us/iis/configuration/system.webserver/security/).
3. In the pivot-accelerator-dll/.vs/Flexmonster Accelerator MVC/config/applicationhost.config file, find the anonymousAuthentication element and set its enabled attribute to true. Then do the same for the windowsAuthentication element:

   ```
   <authentication>
     <anonymousAuthentication enabled="true" userName="" />
     <!-- Other configs -->
   ```

```
  <windowsAuthentication enabled="true">
     <!-- Other configs -->
  </windowsAuthentication>
</authentication>
```

Note that the .vs/ folder appears after opening the project in Visual Studio, and it is hidden by default. See the Microsoft documentation (https://support.microsoft.com/en-us/windows/show-hidden-files-0320fe58-0117-fd59-6851-9b7f9840fdb2) for guidance on displaying hidden folders.

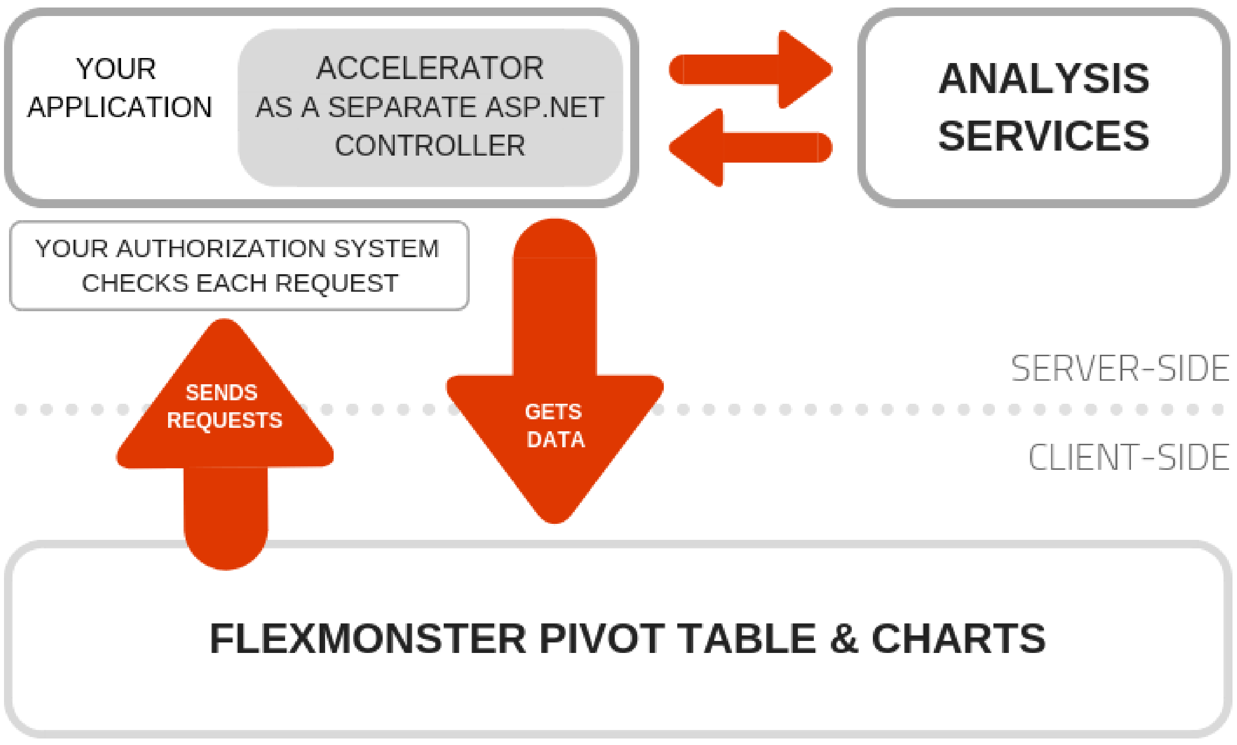
4. Set the dataSource.withCredentials (https://www.flexmonster.com/api/data-source-object/#withCredentials) property to true in your .cshtml page (e.g., Index.cshtml (https://github.com/flexmonster/pivot-accelerator-dll/blob/windows-authentication/Flexmonster%20Accelerator%20MVC/Views/Home/Index.cshtml#L28)):

```
dataSource: {
    dataSourceType: "microsoft analysis services",
    proxyUrl: "/api/accelerator/",
    catalog: "Your Catalog",
    cube: "Your Cube",
    binary: true,
    withCredentials: true
},
```

Now you can use Windows authentication in your Accelerator DLL project.

3. Using custom authorization

If you already have an ASP.NET portal that handles users and an authorization process, the most convenient option is to embed the Accelerator into that system. For this purpose, we recommend referencing the Accelerator as a DLL and integrating a Web API endpoint. Endpoint access is fully controlled by the ASP.NET portal so you can manage security in any way you want. The overall process is described in the diagram below. For more details regarding referencing the Accelerator as a DLL please read our tutorial (/doc/referencing-accelerator-as-a-dll/).


YOUR APPLICATION
ACCELERATOR AS A SEPARATE ASP.NET CONTROLLER
ANALYSIS SERVICES
YOUR AUTHORIZATION SYSTEM CHECKS EACH REQUEST
SENDS REQUESTS
GETS DATA
SERVER-SIDE
CLIENT-SIDE
FLEXMONSTER PIVOT TABLE & CHARTS

What’s next?

You may be interested in the following articles:

- Configuring a secure HTTPS connection (https://www.flexmonster.com/doc/configuring-secure-https-connection/)

4.7.5. SSAS - Configuring a secure HTTPS connection

Overview

All data that is sent by HTTP is not encrypted and can be inspected. For this reason, we added an option to enable HTTPS for the Accelerator. HTTPS encrypts all data that is sent from the client to the Accelerator and vice versa. Follow these steps to configure a secure HTTPS connection for your setup.

Requirements

- Flexmonster Pivot version 2.206 or higher
- Microsoft Analysis Services installed and configured
- Microsoft .NET Framework 4 or higher
- **A valid and trusted SSL certificate**

Step 1. Import the certificate to the certificate store

Skip this step if you have already imported the certificate to your machine’s certificate store. Otherwise, open Windows PowerShell and run the following command:

```
Import-PfxCertificate -FilePath <cert.pfx> -CertStoreLocation Cert:\LocalMachine\My
```

Where:

- <cert.pfx> – the path to your certificate. We recommend using PFX certificates.

Learn more about the Import-PfxCertificate command (https://docs.microsoft.com/en-us/powershell/module/pkiclient/import-pfxcertificate?view=win10-ps).

Once the certificate is imported, you should see the following message in the console:

```
Thumbprint             Subject
----------             -------
XXXXXXXXXXXXXXXXXXXXX  CN=example.com
```

Notice Thumbprint — it is your certificate's SHA-1 hash represented as a hex string. You will need it in the next step.

Step 2. Register the server certificate

On an elevated console ("Run as administrator"), register the server certificate by running this command:

```
netsh http add sslcert ipport=0.0.0.0:<port> certhash=<certHash> appid=<app-guid>
```

Where:

- <port> – the listening port (50005 by default). The IP address 0.0.0.0 matches any IP address for the local machine. Ensure that the port you specify in the <port> parameter is not being used by other servers. Remember to set the same port as the PORT parameter in flexmonster.config.
- <certHash> – the certificate's SHA-1 hash represented as a hex string. It is the same as your certificate's thumbprint from the previous step. Verify that the <certHash> you specify in this command matches the hash of your certificate file.
- <app-guid> – specify a random GUID (formatted like this '{00000000-0000-0000-0000-000000000000}'), used to identify the owning application. Use single quotes and curly braces ('{}') in the appid parameter like this:

  ```
  netsh http add sslcert ipport=0.0.0.0:443 certhash=XXX appid='{YYY}'
  ```

If any errors occur, verify that the SSL certificate is properly installed in the personal store and has an assigned private key. To see all installed SSL certificates, use this command:

```
netsh http show sslcert
```

Step 3. Enable HTTPS in Flexmonster Accelerator

After you have registered the certificate, enable HTTPS in the Accelerator’s config. Open flexmonster.config and modify/add the HTTPS parameter:

```
HTTPS=true
```

Available values for the HTTPS parameter are true or false. By default HTTPS is disabled (false).

Flexmonster Accelerator is now ready to be launched. Just run flexmonster-proxy-ssas.exe with administrator privileges.

You can check if the Accelerator is up and running by navigating to its URL in the browser (https://localhost:50005 (https://localhost:50005/) by default).

Step 4. Configure Flexmonster Pivot

Now it’s time to configure the client – Flexmonster Pivot Table & Charts. Let’s create a minimal configuration using the JavaScript API (replace proxyUrl, catalog, and cube parameters with your specific values):

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "microsoft analysis services",
            proxyUrl: "https://localhost:50005",
            catalog: "Adventure Works DW Standard Edition",
            cube: "Adventure Works",
            binary: true
        }
    }
});
```

Note that proxyUrl now uses https://. That means that the data is protected and encrypted with your SSL certificate.

What’s next?

You may be interested in the following articles:

- Configuring the authentication process (https://www.flexmonster.com/doc/configuring-authentication-process/)

4.7.6. SSAS - Troubleshooting

Error: Your current version of Flexmonster Pivot Table & Charts is not compatible with Flexmonster Accelerator. Please update the component to the minimum required version: X.XXX

This error means that the versions of Flexmonster Accelerator and Flexmonster Pivot are not compatible. To fix this error it is recommended to update both products to the latest available version. Refer to the “How to update” guide (/doc/updating-to-the-latest-version/) for more details.

Error: Your current version of Flexmonster Accelerator is not compatible with Flexmonster Pivot Table & Charts. Please update the Accelerator to the minimum required version: X.XXX

This error means that the versions of Flexmonster Accelerator and Flexmonster Pivot are not compatible. To fix this error it is recommended to update both products to the latest available version. Refer to the “How to update” guide (/doc/updating-to-the-latest-version/) for more details.

4.8. Custom data source API

4.8.1. Introduction to the custom data source API

This section illustrates how to build a report based on your implementation of the Flexmonster custom data source API – our custom communication protocol that allows you to retrieve already aggregated data from a server to Flexmonster Pivot.

The server is responsible for fetching data from a data source, as well as processing and aggregating it. Then the data is passed to Flexmonster Pivot in a ready-to-show format.

Advantages of the Flexmonster custom data source API:

- **More data sources**. Flexmonster retrieves the data from the server in a pre-processed unified format thus delegating the data querying and processing to the server. As a result, you can load the data from any data source, even if it is not supported directly by Flexmonster.
- **Reduced browser memory usage.** All the calculations are performed on the server side and Flexmonster loads only the required part of the data for the report.
- **Larger data amounts.** Since only a subset of the data is loaded into the browser, Flexmonster Pivot can visualize reports for larger data volumes.
- **Full control over data processing**. You can add any additional data processing in your server-side implementation.

To start using the Flexmonster custom data source API, you need to implement the API request handling on your server. We have prepared two sample servers where the handling of these requests is already implemented:

- A quick overview of a sample Node.js server (/doc/pivot-table-with-node-js-server/)
- A quick overview of a sample .NET Core server (/doc/pivot-table-with-dot-net-core-server/)

The following guide will walk you through implementing your own server:
Implementing the custom data source API server. (/doc/implement-custom-data-source-api/)

4.8.2. A quick overview of a sample Node.js server

We have prepared a sample Node.js server that implements the Flexmonster custom data source API. It is available in the server-nodejs/ folder in the api-data-source GitHub repository (https://github.com/flexmonster/api-data-source).

All requests from Flexmonster Pivot Table are handled by the http://localhost:3400/api/cube endpoint. Raw data is stored in JSON format in the server-nodejs/data/ folder. The file name matches the index property of the dataSource configuration object.

Download the .zip archive with the sample project or clone it from GitHub (https://github.com/flexmonster/api-data-source) with the following command:

```
git clone https://github.com/flexmonster/api-data-source
cd api-data-source
```

To start the server, run the following commands in a console:

```
cd server-nodejs
npm install
npm start
```

On the client side (see /client/index.html), the report is configured as follows:

```
new Flexmonster({
    container: "pivotContainer",
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:3400/api/cube",
            index: "fm-product-sales"
        }
    }
});
```

What’s next?

You may be interested in the following articles:

- Implementing the custom data source API server (/doc/implement-custom-data-source-api/)
- Implementing filters (https://www.flexmonster.com/doc/implementing-filters/)
- Returning data for the drill-through view (/doc/return-data-for-drill-through/)
- Supporting more aggregation functions (/doc/support-more-aggregations/)
- A quick overview of a sample .NET Core server (/doc/pivot-table-with-dot-net-core-server/)

4.8.3. A quick overview of a sample .NET Core server

For a quick start in using the Flexmonster custom data source API (our custom communication protocol), we have prepared a sample .NET Core server that implements it. The sample .NET Core server allows loading data from

CSV, JSON, as well as from several databases.

Prerequisites

To run the sample .NET Core server, you need Microsoft .NET Core 3.0 or 3.1. Get it here (https://dotnet.microsoft.com/download/dotnet/3.1) if it's not already installed on your machine.

To start working with the sample .NET Core server, follow these guides:

- Download the sample .NET Core server (#download)
- Available configurations (#configs)
- Connect to the data source (#data-sources)
- Run the sample .NET Core server (#run)
- Configure the report (#report)

Download the sample .NET Core server

To try our sample .NET Core server, download the .zip archive with the sample project or clone it from GitHub (https://github.com/flexmonster/api-data-source) with the following command:

```
git clone https://github.com/flexmonster/api-data-source
cd api-data-source
```

The sample .NET Core server is in the server-dotnetcore/ folder.

All requests from Flexmonster Pivot Table are handled by the http://localhost:3400/api/cube endpoint. Raw data is stored in JSON and CSV formats in the data/ folder.

Available configurations

The sample .NET Core server can be configured in the appsettings.json file which contains the following properties:

- "DataSources" – Array of objects. Configures the data sources. Each object has the following properties:
    - "Type" – String. The type of data source: "json", "csv", or "database".
    - "DatabaseType" (optional) – String. The type of the database: "mysql", "mssql", "postgresql", or "oracle". Only for the "database" data source type.
    - "ConnectionString" (optional) – String. A connection string for the database. Only for the "database" data source type.
    - "Indexes" – Object. Contains a list of datasets. Each dataset is represented by a "key": "value" pair, where "key" is the dataset name, and "value" is an object with the following properties:
        - "Path" (optional) – String. The path to the file with data. Only for "json" and "csv" data source types.
        - "Query" (optional) – String. The query to execute (e.g., "SELECT * FROM tablename"). Only for the "database" data source type.
        - "Delimiter" (optional) – String. Defines the fields separator to split each CSV row. Only for the "csv" data source type. *Default value: ","*.
- "DataStorageOptions" (optional) – Object. Configures the options for data storage. It has the following parameters:
    - "DataRefreshTime" (optional) – Number. Defines how often the data is reloaded from a file or a database. The refresh time is set in minutes. If left unspecified, the data will not be reloaded.

Connect to the data source

The sample .NET Core server configurations vary depending on the data source type. See the following guides to connect the sample .NET Core server to a data source:

- Connecting to JSON (#json)
- Connecting to CSV (#csv)
- Connecting to databases (#databases)

Connecting to JSON

The sample .NET Core server supports only a specific JSON format – an array of objects, where each object is an unordered set of "key": "value" pairs. Here is an example:

```
[
   {
       "Color" : "green",
       "Country" : "Canada",
       "State" : "Ontario",
       "City" : "Toronto",
       "Price" : 174,
       "Quantity" : 22
   },
   ...
]
```

To connect to a JSON data source with the sample .NET Core server, specify the "Type" and "Indexes" properties in the appsettings.json file. For example:

```
"DataSources": [
    {
        "Type": "json",
        "Indexes": {
            "index_json": {
                "Path": "./data/data.json"
            }
        }
    }
],
```

"index_json" is a dataset identifier. It will be used to configure the data source on the client side. Additional indexes can be specified like this:

```
{
    "Type": "json",
    "Indexes": {
        "index_json": {
            "Path": "./data/data.json"
        },
        "another_index_json": {
            "Path": "./data/another_data.json"
        }
    }
}
```

To start the sample .NET Core Server, refer to the Run the sample .NET Core server (#run) guide.

To see how the connection with the sample .NET Core server is configured in the component, refer to the Configure the report (#report) section.

Connecting to CSV

To connect to a CSV data source with the sample .NET Core server, specify the "Type" and "Indexes" properties in the appsettings.json file. For example:

```
"DataSources": [
    {
        "Type": "csv",
        "Indexes": {
            "index_csv": {
                "Path": "./data/data.csv"
            }
        }
    }
],
```

"index_csv" is a dataset identifier. It will be used to configure the data source on the client side. Additional indexes can be specified like this:

```
{
    "Type": "csv",
    "Indexes": {
        "index_csv": {
            "Path": "./data/data.csv"
        },
        "another_index_csv": {
            "Path": "./data/another_data.csv"
        }
    }
}
```

If CSV fields are not separated by "," but by another character, the "Delimiter" parameter should be specified:

```
"index_csv": {
    "Path": "./data/data.csv",
    "Delimiter": ";"
}
```

To start the sample .NET Core Server, refer to the Run the sample .NET Core server (#run) guide.

To see how the connection with the sample .NET Core server is configured in the component, refer to the Configure the report (#report) section.

Connecting to databases

The sample .NET Core server supports MySQL, Microsoft SQL Server, PostgreSQL, Oracle, and Microsoft Azure SQL databases.

To connect to a database with the sample .NET Core server, specify the "Type", "DatabaseType", "ConnectionString", and "Indexes" properties in the appsettings.json file. For example:

```
{
    "DataSources": [
        {
            "Type": "database",
            "DatabaseType": "mysql"
            "ConnectionString": "Server=localhost;Port=3306;Uid=root;Pwd=password;Da
tabase=database_name",
            "Indexes": {
                "index_database": {
                    "Query": "SELECT * FROM tablename"
                }
            }
        }
    ]
}
```

"index_database" is a dataset identifier. It will be used to configure the data source on the client side.

"ConnectionString" is a connection string for the database. Here are some example connection strings for each supported database type:

- MySQL (https://www.connectionstrings.com/mysql-connector-net-mysqlconnection/): "Server=localhost;Port=3306;Uid=;Pwd=;Database= "
- Microsoft SQL Server (https://www.connectionstrings.com/sqlconnection/): "Server=(localdb)\\MSSQLLocalDB;Uid=;Pwd=;Database= "
- PostgreSQL (https://www.connectionstrings.com/npgsql/): "Server=localhost;Port=5432;Uid=;Pwd=;Database= "
- Oracle (https://www.connectionstrings.com/oracle-data-provider-for-net-odp-net/): "Data Source=ORCL;User Id=;Password=;"
- Microsoft Azure SQL (https://www.connectionstrings.com/azure-sql-database/): Server=tcp:myserver.database.windows.net,1433;Database= ;User ID=;Password=;Trusted_Connection=False;Encrypt=True; (to connect to Microsoft Azure SQL, set the "DatabaseType" to "mssql")

To start the sample .NET Core Server, refer to the Run the sample .NET Core server (#run) guide.

To see how the connection with the sample .NET Core server is configured in the component, refer to the Configure the report (#report) section.

Run the sample .NET Core server

To start the server, run the following commands in a console:

```
cd server-dotnetcore
dotnet restore
dotnet run
```

As soon as you start the sample .NET Core server, it automatically preloads the data specified in the "Indexes" property. Thus, when Flexmonster Pivot requests the data, the server responds with the already preloaded data.

The preloaded data is kept in the server's RAM, so the number of indexes you can specify is limited by the amount of RAM available to the server.

Configure the report

On the client side, the report should be configured as follows:

```
new Flexmonster({
    container: "pivotContainer",
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:3400/api/cube",
            index: "index-json"
        }
    }
});
```

index must match the name of the index defined when configuring the data source (e.g., "index_json").

When Flexmonster requests the data, the sample .NET Core server sends a response and then caches it. If the component sends the same request once again, the server responds with the data from its cache.

The cache is fully cleared only when the server is restarted, although it has a memory limit: when the limit is reached, and a new response can't be cached, the .NET Core server deletes one of the previously cached responses from the cache.

What's next?

You may be interested in the following articles:

- Implementing the custom data source API server (/doc/implement-custom-data-source-api/)
- Supporting more aggregation functions (https://www.flexmonster.com/doc/support-more-aggregations/)
- Implementing filters (https://www.flexmonster.com/doc/implementing-filters/)
- Returning data for the drill-through view (/doc/return-data-for-drill-through/)
- A quick overview of a sample Node.js server (/doc/pivot-table-with-node-js-server/)

4.8.4. Implement your own server

4.8.4.1. Implementing the custom data source API server

This guide will help you implement your own custom data source API server. To configure your server so that it can exchange data with Flexmonster, follow these steps:

- Step 1. Set up Flexmonster (#step1)
- Step 2. Configure the connection in the Flexmonster report (#step2)
- Step 3. Create endpoints to handle POST requests (#step3)
- Step 4. Handle the first (handshake) request (#step4)
- Step 5. Handle the request for the data structure (#step5)
- Step 6. Handle requests for members (#step6)
- Step 7. Handle requests for aggregated data (#step7)
- Step 8. Test your custom data source API server (#step8)

Step 1. Set up Flexmonster

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Configure the connection in the Flexmonster report

In report.dataSource, define these parameters to connect to your custom data source API:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "api",
            url: "http://localhost:3400/api/cube",
            index: "data-set-123"
        }
    }
});
```

Here, url is the base URL to your API endpoints and index is the identifier of your data set. index will be sent with every request.

At this step, since the back end isn’t configured yet, you won’t see any data in the pivot table if you open it in a browser.

In the next steps, you will find out how to pass the data from your server to Flexmonster using the custom data source API.

Step 3. Create endpoints to handle POST requests

Flexmonster sends POST requests to the API endpoints using the JSON format. After receiving the responses from the server, it visualizes the data in the pivot table or pivot charts. The first step in the API implementation is to create endpoints on your server to handle these POST requests.

If Flexmonster Pivot is running on a different server, enable CORS.

All requests have the type property in the request body. There are 4 types of requests that can be distinguished by the URL path and type value:

- <url>/handshake – The first (handshake) request to establish a connection between the client and server sides.
- <url>/fields – Request for all fields with their types (i.e., meta-object or schema).
- <url>/members – Request for all members of the field.
- <url>/select – Request for the data.

The value of type will always be the same as the endpoint name, e.g., when a request is sent to <url>/fields, the value of type is "fields".

We also recommend that you check our sample Node.js server (/doc/pivot-table-with-node-js-server/) or sample .NET Core server (/doc/pivot-table-with-dot-net-core-server/) that implements Flexmonster's custom data source API for an example implementation.

Step 4. Handle the /handshake request

After the connection is configured, Flexmonster sends the /handshake request (https://www.flexmonster.com/api/handshake-request) to <url>/handshake. It is used to establish a connection between the client and server sides and exchange some basic information. The front end sends the version of the custom data source API that it implements. Then, Flexmonster Pivot expects the version of the custom data source API implemented by the back end in response.

The /handshake request allows verifying version compatibility. If the server sends the version of the custom data source API in response to the /handshake request, the component can check whether the server and the component implement the same version of the custom data source API.

To receive notifications about version compatibility, respond to the /handshake request with the implemented version of the custom data source API:

```
const API_VERSION = "2.8.5";

cube.post("/handshake", async (req, res) => {
    try {
        res.json({ version: API_VERSION });
    } catch (err) {
        handleError(err, res);
    }
});
```

The /handshake request is optional. If the server does not implement it, Flexmonster will proceed to the next request. However, we recommend handling the /handshake request.

Step 5. Handle the request for the data structure

The next Flexmonster's request is the /fields request that is sent to <url>/fields. Read more details about the /fields request (https://www.flexmonster.com/api/fields-request) in the documentation and implement a response to it on your server.

The custom data source API supports 3 field types: "string", "number", and "date". Note that at least one aggregation has to be supported by the server side for at least one field. For example, a field in the response can have "aggregations": ["sum"] defined:

```
{
    "uniqueName": "Quantity",
    "type": "number",
    "aggregations": ["sum"]
}
```

This means that the back end will provide aggregated data for this field and it can be selected as a measure in Flexmonster Pivot.

When Flexmonster Pivot successfully receives the response to the /fields request, the Field List with all available fields is shown. To see it, open the HTML page in a browser.

From now on, the component configured in step 2 (#step2) should show you the data.

Step 6. Handle requests for members

The next request to handle is the request for the field's members that is sent to <url>/members.

Read more details about the /members request (https://www.flexmonster.com/api/members-request) in the documentation and implement a response on your server.

Now in the Field List, you will be able to select a string field for rows or for columns and retrieve its members.

For the custom data source API, date members should be passed to Flexmonster in the Unix timestamp (https://www.unixtimestamp.com/) format to be recognized correctly. For example, the date "2016-02-07" is "1454803200" in the form of a Unix timestamp.

Step 7. Handle requests for aggregated data

When a field is selected for rows and/or columns, and a numeric field is selected for measures in the Field List, the /select request (https://www.flexmonster.com/api/select-request-for-pivot-table) is sent to the endpoint <url>/select.

To handle the /select request, your server must implement at least one aggregation function. It is easiest to start with one aggregation (e.g., "sum") and extend the list of supported aggregations later.

This is the time to handle the query.aggs part of the request (the /select request can also have query.filter and query.fields, but they can be skipped for now):

```
{
    "type": "select"
    "index": string,
    "query": {
        "aggs": {
            "values"[]: {
                "field": FieldObject,
                "func": string
            },
            "by": {
                "rows": FieldObject[],
                "cols": FieldObject[]
            }
        }
    }
}
```

When Flexmonster successfully receives the response to a /select request, the pivot table is shown.

Step 8. Test your custom data source API server

As a finishing touch, you can check if your server handles custom data source API requests as expected. For this purpose, we created a test suite that covers basic use cases. To learn more about server testing, see this guide (https://www.flexmonster.com/doc/test-custom-data-source-api-server/).

What’s next?

You may be interested in the following articles:

- Implementing filters (/doc/implementing-filters/)
- Returning data for the drill-through view (/doc/return-data-for-drill-through/)
- Supporting multilevel hierarchies (https://www.flexmonster.com/doc/supporting-multilevel-hierarchies/)
- Supporting more aggregation functions (/doc/support-more-aggregations/)

4.8.4.2. Implementing filters

The UI filters are disabled for the custom data source API connection in Flexmonster by default. You can choose which filters to enable in Flexmonster and support on the back end. This filter configuration can be done in the schema (/api/fields-request) and the filtering itself should be implemented in the query.filter part of the /select request (/api/select-request-for-pivot-table).

The guide focuses on how to configure filters. Filters can be:

- Turned on all at once (#enable-all-filters)
- Configured separately for fields of a certain type (#fields-of-certain-type)
- Defined for each field individually (#individual-fields)

Available filter configurations

There are three properties in the /fields request (/api/fields-request/) through which filters can be configured:

- members (optional) – Boolean. Configures an include/exclude members filter. If true, the members filter is enabled in Flexmonster Pivot.
- query (optional) – Boolean|Array of strings. Configures a conditional filter on members. This filter can be

turned on either by setting the property to true or by specifying an array of supported conditions.
If this property is set to true, it enables all the conditions that exist in Flexmonster Pivot for the members filter. See the list of supported conditions for “string” (#string), “number” (#number), and “date” (#date) field types.

- valueQuery (optional) – Boolean|Array of strings. Configures a conditional filter on values. This filter can be turned on either by setting the property to true or by specifying an array of supported conditions.
  If this property is set to true, it enables all the conditions that exist in Flexmonster for the values filter. See the list of supported conditions (#values).

These configurations can be applied to all fields (#enable-all-filters), to fields of a certain type (#fields-of-certain-type), and/or to certain fields (#individual-fields).

Turn on all filters at once

Turning on all filters at once is the simplest way to enable filtering. To enable all available filters, set the root filters property of the /fields request (/api/fields-request/) to true:

```
"filters": true,
```

This approach requires implementing all the available filters in Flexmonster Pivot (see the list of supported filters (#supported-conditions)).

Configure filters for fields of a certain type

It is possible to specify the supported filtering conditions for fields of a certain type. This can be done in the root filters property of the /fields request (/api/fields-request/).

Filters for any field type

Filters that are common for all the field types can be defined by specifying the filters.any property:

```
"filters": {
    "any": {
        "members": true,
        "query": ["equal", "not_equal"],
        "valueQuery": ["top","bottom"]
    }
}
```

Here, the query property can contain only "equal", "not_equal", "between", and "not_between" conditions since these are the only conditions that are common for all the field types.

In the valueQuery property, you can specify which conditions are supported by the filter on values (#values).

Filters for fields of a certain type

The string, number, and date properties allow you to configure filters for fields of a corresponding type. Here is an example of filters configuration for the "string" field type:

```
"filters": {
    "string": {
        "members": true,
        "query": ["equal", "not_equal"],
        "valueQuery": ["top", "bottom"]
    }
}
```

Here, the query property can contain conditions that are supported by the “string” field type (#string).

In the valueQuery property, you can specify which conditions are supported by the filter on values (#values).

Filters for the fields of the "number" and "date" types can be configured similarly. See the list of supported conditions for “number” (#number) and “date” (#date) field types.

Define filters for individual fields

Filters can also be configured for individual fields. This can be done by specifying the fields.filters property:

```
"fields": [{
    //other properties
    "filters": {
        "members": true,
        "query": ["equal", "not_equal"],
        "valueQuery": ["top", "bottom"]
    }
}],
```

In the valueQuery property, you can specify which conditions are supported by the filter on values (#values).

Conditions that can be defined in the query property depend on the field’s type. See the list of supported conditions for “string” (#string), “number” (#number), and “date” (#date) field types.

Supported filtering conditions

Conditions for the filter on values

Conditions for the filter on values are common for all field types. The supported conditions are the following: "top", "bottom", "equal", "not_equal", "greater", "greater_equal", "less", "less_equal", "between", "not_between".

For the members filter, the supported conditions depend on the field type. Have a look at the list of supported conditions for “string” (#string), “number” (#number), and “date” (#date) fields below.

Conditions for the “string” field type

The filter on members supports the following conditions for the "string" field type: "equal", "not_equal", "begin", "not_begin", "end", "not_end", "contain", "not_contain", "greater", "greater_equal", "less", "less_equal", "between", "not_between".

Conditions for the “number” field type

The filter on members supports the following conditions for the "number" field type: "equal", "not_equal", "greater", "greater_equal", "less", "less_equal", "between", "not_between".

Conditions for the “date” field type

The filter on members supports the following conditions for the "date" field type: "equal", "not_equal", "before", "before_equal", "after", "after_equal", "between", "not_between", "last", "current", "next".

After configuring the filters in the schema, the handling of the query.filter part of the /select request (/api/select-request-for-pivot-table/) should be implemented.

Note that the server always receives either "between" or "not_between" in the filtering request when Flexmonster Pivot uses the following date filters:

- "equal"
- "not_equal"
- "last"
- "current"
- "next"

To use these date filters on the client, implement the "between" and "not_between" filters on the server.

What’s next?

You may be interested in the following articles:

- Implementing the custom data source API server (/doc/implement-custom-data-source-api/)
- Returning data for the drill-through view (/doc/return-data-for-drill-through/)
- Supporting more aggregation functions (/doc/support-more-aggregations/)
- Supporting multilevel hierarchies (https://www.flexmonster.com/doc/supporting-multilevel-hierarchies/)

4.8.4.3. Supporting more aggregation functions

Using the custom data source API allows you to decide which aggregations to enable in Flexmonster and support on the back end.

The supported aggregations can be defined in the schema (https://www.flexmonster.com/api/fields-request) for fields of a certain type or each field individually. The aggregations supported by Flexmonster for the custom data source API connection are the following: "sum", "count", "distinctcount", "average", "median", "product", "min", "max", "stdevp", "stdevs", "none", or a custom aggregation (#custom-aggs).

It is possible to define which aggregations are available for which fields in the back end; it is also not necessary to implement all the aggregations supported by Flexmonter.

Built-in front-end aggregations

For the custom data source API, Flexmonster supports the following built-in front-end aggregations: "percent", "percentofcolumn", "percentofrow", "percentofparentcolumntotal", "percentofparentrowtotal", "index", "differenceofcolumn", "differenceofrow", "%differenceofcolumn", "%differenceofrow", "runningtotalsofcolumn", "runningtotalsofrow".

These aggregations will appear in Flexmonster Pivot automatically if at least one aggregation is implemented on the back end (e.g., "sum").

This feature is available only for the fields of the "number" type.

Custom aggregations

Besides aggregations supported by Flexmonster Pivot, you can implement a custom aggregation on your server and define it in the schema (https://www.flexmonster.com/api/fields-request):

```
{
  "aggregations": {
    "number": ["sum", "average", "squareroot"],
    …
  }
}
```

When the custom aggregation is defined in the schema (https://www.flexmonster.com/api/fields-request), you can use it on the client side. In the component, a measure with the custom aggregation applied will look similar to the following: Price (squareroot).

Manage the list of available aggregations in Flexmonster

On the client side, you can customize the list of aggregations available for a specific field with a Mapping Object (/api/mapping-object/). This object has the "aggregations" property, which defines a list of available aggregations. That list can contain both the aggregations supported on the back end (including custom ones) and the built-in front-end aggregations.

To learn more about the "aggregations" property, refer to the Mapping guide (/doc/mapping/#aggregations).

What’s next?

You may be interested in the following articles:

- Implementing the custom data source API server (/doc/implement-custom-data-source-api/)
- Implementing filters (https://www.flexmonster.com/doc/implementing-filters/)
- Supporting multilevel hierarchies (https://www.flexmonster.com/doc/supporting-multilevel-hierarchies/)
- Returning data for the drill-through view (/doc/return-data-for-drill-through/)

4.8.4.4. Supporting multilevel hierarchies

Flexmonster supports creating multilevel hierarchies from non-hierarchical data out of the box. To use this feature for the custom data source API, you need to handle certain request scenarios on your server.

This guide describes how to support multilevel hierarchies on your server and configure them in the component.

Step 1. Implement advanced hierarchical filters

Hierarchies are composed on the client side while the server's task is to filter them. For this purpose, your back end has to implement advanced hierarchical filters.

Flexmonster sends filters in /members (https://www.flexmonster.com/api/members-request/) and /select (https://www.flexmonster.com/api/select-request-for-pivot-table/) requests. The filters' structure depends on the custom data source API version implemented by your server (see how to check your version (#check-api-version)):

Version 2.8.22 or later

If the version sent in the /handshake response (https://www.flexmonster.com/api/handshake-request/#response) is 2.8.22 or later, your server should support a filter structure described by the Filter Group Object (https://www.flexmonster.com/api/filter-group-object/).

Version 2.8.5

In version 2.8.5, the Filter Object (https://www.flexmonster.com/api/filter-object-for-requests/) describes the structure of the filters sent by the component.

The include.filter and exclude.filter properties of the Filter Object (https://www.flexmonster.com/api/filter-object-for-requests/) are responsible for filtering hierarchical data. Therefore, implement them to enable the support of multilevel hierarchies on your server.

Step 2. Notify Flexmonster about implementing advanced filters

When the server implements hierarchical filters, it should inform the component about that.

For this purpose, set the advanced property (https://www.flexmonster.com/api/fields-request/#advanced) of the /fields request to true. The component will ignore multilevel hierarchies if the advanced property is set to false.

Step 3. (optional) Handle the additional /select request

The component sends the additional /select request when the levelName property is defined in the slice (https://www.flexmonster.com/doc/slice/). This request extracts members from the hierarchy level specified in levelName.

If you need the levelName property in your slice, handle the /select request (https://www.flexmonster.com/api/select-request-for-pivot-table/) for loading the required hierarchy levels.

Step 4. Configure the multilevel hierarchies

There are two ways to compose multilevel hierarchies:

On the client side

The Mapping Object (https://www.flexmonster.com/doc/mapping/) is responsible for grouping data into hierarchies. Specify hierarchy and parent properties to structure the data.

The example below illustrates how to compose multilevel hierarchies in the report. Here, we create the "Item" hierarchy with the "Category" field as a first level and the "Color" field as a second level:

```
report: {
    dataSource: {
        type: "api",
        url: "your_url",
        index: "your_index",
        mapping: {
            "Category": {
                type: "string",
                hierarchy: "Item"
            },
            "Color": {
                type: "string",
                hierarchy: "Item",
                parent: "Category"
            }
        }
    },
    slice: {
        // your slice
    }
}
```

See a live example on JSFiddle (https://jsfiddle.net/flexmonster/o4jywt38/).

On the server side

As an alternative to the Mapping Object, you can configure hierarchies right in the response to the /fields request (https://www.flexmonster.com/api/fields-request/). To do so, specify fields.hierarchy and fields.parent properties.

Here is an example of a /fields response where the "Category" and "Color" fields are grouped under the "Item" hierarchy (with "Category" as the top level of the hierarchy):

```
{
    "fields":[
        {
            "uniqueName": "Category",
            "type": "string",
            "hierarchy": "Item"
        },
        {
            "uniqueName": "Color",
            "type": "string",
            "hierarchy": "Item",
            "parent": "Category"
        },
        ...
    ],
    "aggregations":{
        ...
    },
    "filters":{
        ...
    }
}
```

Now run your project and open the web page with Flexmonster – multilevel hierarchies should appear in the component.

Check your version of the custom data source API

To find out which version of the custom data source API your server implements, you can use the /handshake request (https://www.flexmonster.com/api/handshake-request/). There are two possible options:

- /handshake is implemented on your server.
  If the server handles this request, a version sent in the response is your custom data source API version.
- /handshake is not implemented on your server.
  In this case, Flexmonster considers 2.8.5 to be your version of the custom data source API.

What’s next?

You may be interested in the following articles:

- Implementing the custom data source API server (https://www.flexmonster.com/doc/implement-custom-data-source-api/)
- Implementing filters (https://www.flexmonster.com/doc/implementing-filters/)
- Returning data for the drill-through view (https://www.flexmonster.com/doc/return-data-for-drill-through/)
- Supporting more aggregation functions (https://www.flexmonster.com/doc/support-more-aggregations/)

4.8.4.5. Returning data for the drill-through view

The /select requests are also responsible for retrieving data for the drill-through view (https://www.flexmonster.com/api/select-request-for-drill-through-view).

At this step, the handling of the query.fields part of the /select request should be implemented.

What’s next?

You may be interested in the following articles:

- Implementing the custom data source API server (/doc/implement-custom-data-source-api/)
- Implementing filters (/doc/implementing-filters/)
- Supporting more aggregation functions (/doc/support-more-aggregations/)
- Supporting multilevel hierarchies (https://www.flexmonster.com/doc/supporting-multilevel-hierarchies/)

4.8.4.6. Testing your custom data source API server

To check whether your server handles the custom data source API requests (https://www.flexmonster.com/api/all-requests/) as expected, you can use our test suite that covers basic use cases.

Our testing program works for any server implementing the custom data source API. It sends requests to your back end and expects a specific response in return.

Testing your server requires two stages:

1. Prepare your server (#prepare-server)
2. Run the tests (#run-tests)

Prepare your server

To pass our tests, your server should have an appropriately configured test index:

- As a test dataset, use either data.json (https://github.com/flexmonster/api-data-source/blob/master/server-nodejs/data/data.json) or data.csv (https://github.com/flexmonster/api-data-source/blob/master/server-dotnetcore/data/data.csv) files.
- Our tests use the data index in requests to the server. Therefore, create an index with this name for the test data.

After configuring the index, run your server.

Run the tests

Complete the steps below to download, configure, and run our tests.

Step 1. You can find the test suite on our GitHub (https://github.com/flexmonster/api-data-source/tree/master/tests). Download the repository as a .zip archive or clone it with the following command:

```
git clone https://github.com/flexmonster/api-data-source
cd api-data-source/tests
```

Step 2. Install npm dependencies described in package.json:

```
npm install
```

Step 3. The next step is to adjust the tests to your server. Open the tests/config.json file and specify the following properties:

- "url" — String. The path to your API endpoints (e.g., http://localhost:3400/api/cube).
- "emptyValue" — Any value. Defines how your server should treat null or undefined values (e.g., as empty strings).
- "valueFilters" — Boolean. Indicates whether to run tests for value filters (https://www.flexmonster.com/doc/implementing-filters/#valueQuery). Set this property to false if your server does not implement them.
  Note that these tests will work if your server:
  - Implements version 2.8.5.
  - Implements version 2.8.22 without multilevel hierarchy support (https://www.flexmonster.com/doc/supporting-multilevel-hierarchies/).
- "hierarchy" — Boolean. Indicates whether to run tests for multilevel hierarchies

(https://www.flexmonster.com/doc/supporting-multilevel-hierarchies/). Set this property to false if your server does not implement them.
Note that these tests will work if your server implements version 2.8.5 of the custom data source API. See how to check your version (https://www.flexmonster.com/doc/supporting-multilevel-hierarchies/#!check-api-version).

Step 4. Run the test suite with the following command:

```
npm test
```

You will see test results in the console.

Test overview

Let’s have a look at the files you can find in the tests/test/ folder:

- HandshakeRequestSpec.js (https://github.com/flexmonster/api-data-source/blob/master/tests/test/HandshakeRequestSpec.js) – Tests the /handshake request (https://www.flexmonster.com/api/handshake-request/) implementation.
- FieldsRequestSpec.js (https://github.com/flexmonster/api-data-source/blob/master/tests/test/FieldsRequestSpec.js) – Tests the /fields request (https://www.flexmonster.com/api/fields-request/) implementation.
- MembersRequestSpec.js (https://github.com/flexmonster/api-data-source/blob/master/tests/test/MembersRequestSpec.js) – Tests the /members request (https://www.flexmonster.com/api/members-request/) implementation.
- SelectRequestSpec.js (https://github.com/flexmonster/api-data-source/blob/master/tests/test/SelectRequestSpec.js) – Tests the /select request (https://www.flexmonster.com/api/select-request-for-pivot-table/) implementation.
- SelectRequestFlatSpec.js (https://github.com/flexmonster/api-data-source/blob/master/tests/test/SelectRequestFlatSpec.js) – Tests the /select request for the flat table (https://www.flexmonster.com/api/select-request-for-flat-table/) implementation.
- SelectRequestDrillThroughSpec.js (https://github.com/flexmonster/api-data-source/blob/master/tests/test/SelectRequestDrillThroughSpec.js) – Tests the /select request for the drill-through view (https://www.flexmonster.com/api/select-request-for-drill-through-view/) implementation.

What’s next?

You may be interested in the following articles:

- How to configure which data subset is shown (https://www.flexmonster.com/doc/slice/)
- How to sort the data (https://www.flexmonster.com/doc/custom-sorting/)
- How to customize the component (https://www.flexmonster.com/doc/available-tutorials-customizing/)

4.9. Elasticsearch

4.9.1. Connecting to Elasticsearch

This article illustrates how to build a report based on an Elasticsearch data source.

Requirements

- Flexmonster Pivot version 2.7.0 or higher
- Elasticsearch version 6.x or 7.x

Step 1: Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2: Enable cross-origin resource sharing (CORS)

By default, the browser prevents JavaScript from making requests across domain boundaries. CORS allows web applications to make cross-domain requests. To enable CORS, open elasticsearch.yml and add the following lines:

```
http.cors.enabled: true
http.cors.allow-origin: "*"
http.cors.allow-credentials: true
http.cors.allow-headers: "X-Requested-With, Content-Type, Content-
Length, Authorization"
```

To allow a connection to the Elasticsearch server from machines other than localhost, open an appropriate port in the firewall. The default port is 9200, but it may be different depending on your configuration in the elasticsearch.yml file.

Step 3: Configure the report with your own data

Now it’s time to configure the pivot table on the web page. Let’s create a minimal report for this (replace the node and index parameters with your specific values):

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        "dataSource": {
            "type": "elasticsearch",
            /* the host for the connection */
            "node": "https://olap.flexmonster.com:9200",
            /* the name of Elasticsearch index to connect */
            "index": "fm-product-sales"
        }
    }
});
```

Launch the web page from a browser — there you go! A pivot table is embedded into your project. Check out the example on JSFiddle (https://jsfiddle.net/flexmonster/xfzws81u/).

What’s next?

You may be interested in the following articles:

- Configuring the mapping (/doc/configuring-the-mapping/)

4.9.2. Configuring the mapping

This tutorial explains how to define a mapping object for an Elasticsearch index in a report and which settings this mapping object supports.

A mapping object can have the following properties:

- caption (optional) – String. Overrides the default name of the field.
- visible (optional) – Boolean. When set to false, hides the field from the Field List.
- interval (optional) – String. Used for the date histogram (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html). Check out the supported time units (https://www.elastic.co/guide/en/elasticsearch/reference/current/common-options.html#time-units).
- time_zone (optional) – String. Used for the date histogram (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html). You can specify time zones as either an ISO 8601 UTC offset (e.g., +01:00 or -08:00) or as a time zone ID as specified in the IANA time zone database, such as America/Los_Angeles. Check out this example (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html#_time_zone_2).
- format (optional) – String. Used for the date histogram (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html). Check out the supported date format/patterns (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-daterange-aggregation.html#date-format-pattern).
  If the datePattern (https://www.flexmonster.com/api/options-object/#datePattern) option is defined, format will override its value for the field.
- min_doc_count (optional) – Number. Used for the date histogram (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html). Can be used to show intervals with empty values (min_doc_count: 0). *Default value: 1 (empty intervals are hidden)*.

How to hide unnecessary fields in Elasticsearch

All unnecessary fields can be hidden by setting visible: false:

```
var pivot = new Flexmonster({
  container: "pivotContainer",
  toolbar: true,
  report: {
    dataSource: {
      type: "elasticsearch",
      /* the host for the connection */
      node: "https://olap.flexmonster.com:9200",
      /* the name of Elasticsearch index to connect */
      index: "fm-product-sales",
      /* additional setting to configure index mapping */
      mapping: {
        "@timestamp": {
          visible: false
        },
        "@version": {
          visible: false
        },
        "host": {
          visible: false
        },
        "message": {
          visible: false
        },
        "path": {
          visible: false
        }
      }
    }
  }
});
```

Check out the example on JSFiddle (https://jsfiddle.net/flexmonster/5aqrx4mc/).

How to format dates in Elasticsearch

There are two ways to format dates in Elasticsearch:

- Using the options.datePattern (https://www.flexmonster.com/api/options-object/#datePattern) property – this will apply formatting to all date fields in the dataset.
- Using the mapping.format property – this will apply formatting to a certain field. If options.datePattern is defined, mapping.format will override its value.

options.datePattern

When formatting dates with the options.datePattern property, use date patterns described in the Elasticsearch documentation (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-daterange-aggregation.html#date-format-pattern).

The following example demonstrates how to format dates using the options.datePattern:

```
new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
      dataSource: {
        type: "elasticsearch",
        node: "https://olap.flexmonster.com:9200",
        index: "fm-product-sales"
      },
      options: {
        datePattern: "dd MMMM, yyyy"
      }
    }
});
```

See the full code on JSFiddle (https://jsfiddle.net/flexmonster/8mfkh3L1/).

mapping.format

When formatting dates with the mapping.format property, use date patterns described in the Elasticsearch documentation (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-daterange-aggregation.html#date-format-pattern).

The following example demonstrates how to format dates using mapping.format:

```
new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
      dataSource: {
        type: "elasticsearch",
        node: "https://olap.flexmonster.com:9200",
        index: "fm-product-sales",
        mapping: {
          "@timestamp": {
            format: "dd/MM/yyyy"
          }
        }
      }
    }
});
```

Check out the example on JSFiddle (https://jsfiddle.net/flexmonster/uvwnrzL0/).

Formatting dates in the drill-through view

Elasticsearch date patterns (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-daterange-aggregation.html#date-format-pattern) are fully applied to dates in the pivot and compact views, while dates in the drill-through view may remain unformatted.

If you need a date pattern that is applied in all the views similarly, format date fields using patterns supported in both Flexmonster and Elasticsearch:

Supported date patterns

Expand the list of the date patterns

- d – Day of the month. It is represented as a one or two-digit number. For example, 2 or 18.
- dd – Day of the month. It is represented as a two-digit number. For example, 02 or 18.
- M – Month. It is represented as a one or two-digit number. For example, 3 or 11.
- MM – Month. It is represented as a two-digit number. For example, 03.
- MMM – Month. It is represented as a three-letter abbreviation of the name of the month. For example, Mar.
- MMMM – Month. It is represented as the full name of the month. For example, March.
- yy – Year. It is represented as a two-digit number. For example, 16.
- yyyy – Year. It is represented as a four-digit number. For example, 2016.
- h – Hour of the day using the 12-hour format [1 – 12]. It is represented as a one or two-digit number. For example, 1 or 12.
- hh – Hour of the day using the 12-hour format [1 – 12]. It is represented as a two-digit number. For example, 01 or 12.
- H – Hour of the day using the 24-hour format [0 – 23]. It is represented as a one or two-digit number. For example, 0 or 23.
- HH – Hour of the day using the 24-hour format [0 – 23]. It is represented as a two-digit number. For example, 00 or 23.
- k – Hour of the day using the 24-hour format [1 – 24]. It is represented as a one or two-digit number. For example, 1 or 24.
- kk – Hour of the day using the 24-hour format [1 – 24]. It is represented as a two-digit number. For example, 01 or 24.
- m – Minutes [0 – 59]. It is represented as a one or two-digit number. For example, 0 or 59.
- mm – Minutes [0 – 59]. It is represented as a two-digit number. For example, 00 or 59.
- s – Seconds [0 – 59]. It is represented as a one or two-digit number. For example, 0 or 59.
- ss – Seconds [0 – 59]. It is represented as a two-digit number. For example, 00 or 59.

4.10. Pentaho Mondrian

4.10.1. Connecting to Pentaho Mondrian

There are two ways to connect to Pentaho Mondrian using Flexmonster Pivot:

1. Via XMLA (/doc/connecting-to-pentaho-mondrian/#xmla) – an industry standard for data access in analytical systems.
2. Via Flexmonster Accelerator (/doc/connecting-to-pentaho-mondrian/#accelerator) – a special server-side utility developed by Flexmonster.

If you already have configured an XMLA provider it will be easier to start with option #1. If you do not have XMLA or if you need some advanced features (such as increased loading speeds, use of credentials, etc.) – option #2 is the better choice.

Connecting to Pentaho Mondrian via XMLA

XMLA (XML for Analysis) is an industry standard for data access in analytical systems such as OLAP and Data Mining. Follow the steps below to configure a connection to Pentaho Mondrian via XMLA.

Step 1: Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2: Configure XMLA access to the cube

Skip this step if you have XMLA already configured. Otherwise, refer to this article that explains how to configure Mondrian as an XMLA provider (http://mondrian.pentaho.com/documentation/installation.php#5_How_to_configure_Mondrian_as_an_XMLA_provider).

Step 3: Enable cross-origin resource sharing (CORS)

By default, the browser prevents JavaScript from making requests across domain boundaries. CORS allows web applications to make cross-domain requests. Here are some instructions for common Java servers:

- Tomcat – Configuration Reference (CORS Filter) (http://tomcat.apache.org/tomcat-8.0-doc/config/filter.html#CORS_Filter)
- Jetty – Jetty/Feature/Cross Origin Filter (http://wiki.eclipse.org/Jetty/Feature/Cross_Origin_Filter)
- JBOSS – CORS Filter Installation (http://software.dzhuvinov.com/cors-filter-installation.html)

Step 4: Configure the report with your own data

Now it’s time to configure the pivot table on the web page. Let’s create a minimal report for this (replace proxyUrl, dataSourceInfo, catalog, and cube parameters with your specific values):

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "mondrian",

            /* Data Source Info */
            dataSourceInfo: "MondrianFoodMart",

            /* URL to XMLA provider */
            proxyUrl: "http://localhost:8080/mondrian/xmla",

            /* Catalog name */
            catalog: "FoodMart",

            /* Cube name */
            cube: "Sales"
        }
    }
});
```

Launch the web page from a browser — there you go! A pivot table is embedded into your project.

Connecting to Pentaho Mondrian via Flexmonster Accelerator

Flexmonster Accelerator for Pentaho Mondrian cubes is a special server-side proxy that increases data loading speeds from the server to customer’s browser.

When working with OLAP cubes, a browser component communicates with the server via the XMLA protocol. The XMLA protocol is heavy and exchanges a lot of excessive information – it takes too much time and memory to load and process the data.

We replaced the XMLA protocol and use direct requests from the component to the server.

This is our solution to two major problems that many of those who work with big data face:

- We made big data transfer from the server to the browser incredibly fast. Our tool allows you to transfer large multidimensional data super easily and quickly. Reporting becomes more enjoyable and prompt for your end users.
- We greatly reduced the load on the browser memory.

Refer to the Getting started with Flexmonster Accelerator (/doc/getting-started-with-accelerator-mondrian/) guide to find step-by-step instructions.

4.10.2. Mondrian - Getting started with Flexmonster Accelerator

Flexmonster Accelerator for Pentaho Mondrian cubes is a special server-side proxy that increases data loading speeds from the server to customer’s browser.

When working with OLAP cubes, a browser component communicates with the server via the XMLA protocol. The XMLA protocol is heavy and exchanges a lot of excessive information – it takes too much time and memory to load and process the data.

We replaced the XMLA protocol and use direct requests from the component to the server.

This is our solution to two major problems that many of those who work with big data face:

- We made big data transfer from the server to the browser incredibly fast. Our tool allows you to transfer large multidimensional data super easily and quickly. Reporting becomes more enjoyable and prompt for your end users.
- We greatly reduced the load on the browser memory.

Requirements

- Flexmonster Pivot version 2.2 or higher
- A relational database with a Mondrian schema
- Java JRE 1.7+

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
```

```
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Configure Flexmonster Accelerator on the server

The package with the Accelerator contains the following files in the server/ folder:

- flexmonster-proxy-mondrian.jar – a server-side utility that handles connectivity between Pentaho Mondrian and Flexmonster Pivot.
- flexmonster-proxy-mondrian-4.jar – same as flexmonster-proxy-mondrian.jar, but for the newer version of Pentaho Mondrian 4.4.
- flexmonster.config – a file that contains configuration parameters for the utility (connection string, port, etc.).

There are also additional sample files:

- FoodMart.xml – a sample Mondrian schema for a FoodMart database.
- FoodMart-4.xml – same as FoodMart.xml, but for the newer version of Pentaho Mondrian 4.4.
- mysql-connector-java-*.jar – a Java connector for a MySQL database.

Let’s review flexmonster.config file. It contains the following parameters:

- CONNECTION_STRING – (**required**) the connection string for Pentaho Mondrian. Example:

  ```
  Jdbc=jdbc:mysql://localhost:3306/foodmart?user=root&password=password;JdbcDriv
  ers=com.mysql.jdbc.Driver;
  ```

- JDBC_DRIVER_JAR – (**required**) the path to the Java connector (a .jar file) for the database. Example: ./mysql-connector-java-5.1.37.jar.
- JDBC_DRIVER_CLASS – (**required**) the class name of the Java connector. Example: com.mysql.jdbc.Driver.
- CATALOGS_PATH – (optional) a path to the folder that contains the Mondrian schemas. *Default value: ./*.
- PORT – (optional) the port number for the proxy service endpoint. *Default value: 50006*.
- CACHE_MEMORY_LIMIT – (optional) the maximum memory size available for caching (in MB). *Default value: 0 (unlimited)*.
- CACHE_ENABLED – (optional) indicates whether caching is enabled.
  Available since version 2.211. *Default value: true*.
- HTTPS – (optional) indicates whether the HTTPS protocol is enabled. *Default value: false*.

After configuring all the necessary options, the Accelerator is ready to be launched. Just execute the following commands in terminal:

```
# navigate to the folder that contains the Accelerator
cd {path_to_package}/server
# start the Accelerator
java -jar flexmonster-proxy-mondrian.jar
```

If your schema is built for Mondrian 4, use flexmonster-proxy-mondrian-4.jar instead.

Step 3. Open a port for Flexmonster Accelerator in the firewall

If you plan to allow connections to the Accelerator from outside the server, open the appropriate port in the firewall. The default port number is 50006, but it can be changed using the PORT parameter in the flexmonster.config file.

Step 4. Configure Flexmonster Pivot Table and Charts

Now it's time to configure the client – Flexmonster Pivot Table and Charts. Let's create a minimal configuration using the JavaScript API (replace proxyUrl, catalog, and cube parameters with your specific values):

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "mondrian",
            /* URL to Flexmonster Accelerator */
            proxyUrl: "http://localhost:50006",
            /* Data source info */
            dataSourceInfo: "MondrianFoodMart",
            /* Catalog name / Mondrian schema */
            catalog: "FoodMart",
            /* Cube name */ cube: "Sales",
            // Flag to use the Accelerator instead of XMLA protocol
            binary: true
        }
    }
});
```

Launch the web page from a browser — there you go! A pivot table is embedded into your project.

Cache control

Flexmonster Accelerator for Mondrian has two levels of cache:

- Mondrian cache
- Accelerator cache (controlled by the CACHE_ENABLED parameter)

Usually a cache can improve performance greatly. However if the underlying database ever changes, the cache goes out of date. Mondrian cannot deduce when the database is being modified, so we introduce a method to force clear the Mondrian cache.

It works by triggering the "ClearCache" as follows:

```
http://localhost:50006/FlexmonsterProxy/ClearCache (http://localhost:50006/Flexmonst
erProxy/ClearCache)
```

The Accelerator cache can be disabled with the CACHE_ENABLED parameter in flexmonster.config (available since version 2.211):

```
CACHE_ENABLED=false
```

Mondrian configuration

Mondrian has a properties file to allow you to configure how it is executed. It is possible to use any of these properties with the Accelerator.

To do this, create a mondrian.properties file in the same location as flexmonster-proxy-mondrian.jar.

For example:

```
mondrian.rolap.aggregates.ChooseByVolume = false
mondrian.rolap.aggregates.generateSql = false
```

Refer to the full list of Mondrian properties (https://mondrian.pentaho.com/documentation/configuration.php) to find out more.

4.10.3. Configuring Mondrian roles

Overview

Sometimes it’s necessary to limit access to data. To do this you can define an access-control profile, called a role, as part of the Mondrian schema and set this role when establishing a connection with Flexmonster Pivot.

Requirements

- Flexmonster Pivot version 2.210 or higher
- A relational database with a Mondrian schema
- Java JRE 1.7+

Step 1: Configure roles in the Mondrian schema file

Firstly configure roles in the Mondrian schema file. Refer to the Mondrian documentation (https://mondrian.pentaho.com/documentation/schema.php#Access_control) for more details.

Step 2: Launch Flexmonster Accelerator

Start (or restart) Flexmonster Accelerator for Mondrian. Refer to the Accelerator “Getting Started” guide (/doc/getting-started-with-accelerator-mondrian/) for more details.

Step 3: Configure Flexmonster Pivot

Now, let’s specify Mondrian roles in the configuration. Here is a minimal sample created with the JavaScript API (replaceproxyUrl, catalog, cube, and roles parameters with your specific values):

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "mondrian",
            proxyUrl: "localhost:50006",
            dataSourceInfo: "MondrianFoodMart",
            catalog: "FoodMart",
            cube: "Sales",
            binary: true,
            // Mondrian roles
            roles: "California manager"
        }
    }
});
```

4.10.4. Mondrian - Configuring username/password protection

Overview

Flexmonster Pivot Table and Charts supports the use of credentials to restrict access to the data source. They are commonly used for the following purposes:

- Providing credentials to connect to the data source.
- Supporting different roles and access levels for users.

Here are step-by-step instructions on how to configure a secure connection with username/password protection.

Requirements

- Flexmonster Pivot version 2.205 or higher
- A relational database with a Mondrian schema
- Java JRE 1.7+

Step 1: Create a default user for Flexmonster Accelerator

We recommend creating a default user for Flexmonster Accelerator with default privileges. This user will be used to start the Accelerator, check the connection to the data source, and connect anonymous users. Refer to the documentation of your database to find out how to create a new user (e.g. MySQL (https://dev.mysql.com/doc/refman/5.7/en/adding-users.html), Oracle (https://docs.oracle.com/cd/B19306_01/network.102/b14266/admusers.htm)).

Step 2: Configure Flexmonster Accelerator

Open flexmonster.config and specify the user and password in the CONNECTION_STRING parameter. After your edits, CONNECTION_STRING should look like this:

```
CONNECTION_STRING=Jdbc=jdbc:mysql://localhost:3306/foodmart?user=flexmonster&passwor
d=password;JdbcDrivers=com.mysql.jdbc.Driver;
```

The Accelerator is ready to be launched. Just execute the following command in terminal:

```
java -jar flexmonster-proxy-mondrian.jar
```

You can check if the Accelerator is up and running by navigating to its URL in the browser (http://localhost:50006 (http://localhost:50006/) by default).

4.10.5. Mondrian - Configuring a secure HTTPS connection

Overview

All data that is sent by HTTP is not encrypted and can be inspected. This is why we have added an option to enable HTTPS for Flexmonster Accelerator. HTTPS encrypts all data that is sent from the client to the Accelerator and vice versa. Follow these steps to configure a secure HTTPS connection for your setup.

Requirements

- Flexmonster Pivot version 2.205 or higher
- A relational database with a Mondrian schema
- Java JRE 1.7+
- **A valid and trusted SSL certificate**

Step 1: Import the certificate to Java KeyStore

A Java KeyStore (JKS) is a repository of security certificates. JDKs provide a keytool utility to manipulate the keystore.

If you already have the private key in PKCS 12, just navigate to JRE/bin/ folder and execute this command:

```
keytool -importkeyst
ore -destkeystore keystore.jks -srckeystore <private_key> -srcstoretype PKCS12
```

Where <private_key> is the path to .p12 file with the private key (i.e. flexmonster.p12).

It will ask you to enter a new password for the keystore and a password for the private key. It will then generate a keystore.jks file with the imported private key inside. Copy this file to some location, it will be necessary for the next step.

Step 2: Enable HTTPS in Flexmonster Accelerator

After you have the certificate imported, enable HTTPS in the Accelerator's config. Open the flexmonster.config file and modify/add the necessary parameters as follows:

```
HTTPS=true
KEY_STORE_PATH=<keystore_path>
KEY_STORE_PASSWORD=<keystore_password>
KEY_MANAGER_PASSWORD=<private_key_password>
```

Where:

- <keystore_path> – path to the JKS file (i.e. keystore.jks).
- <keystore_password> – password for the keystore.
- <private_key_password> – password for the imported key.

Available values for the HTTPS parameter are true or false. By default HTTPS is disabled (false).

The Accelerator is ready to be launched. Just execute the following command in terminal:

```
java -jar flexmonster-proxy-mondrian.jar
```

You can check if the Accelerator is up and running by navigating to its URL in the browser (https://localhost:50006 by default).

Step 3: Configure Flexmonster Pivot

Now it's time to configure the client – Flexmonster Pivot. Let's create a minimal configuration using the JavaScript API (replace proxyUrl, catalog, and cube parameters with your specific values):

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            type: "mondrian",
            proxyUrl: "https://localhost:50006",
            dataSourceInfo: "MondrianFoodMart",
            catalog: "FoodMart",
            cube: "Sales",
            binary: true
        }
    }
});
```

Notice that proxyUrl now contains https://. That means that the data is protected and encrypted with your SSL certificate.

4.10.6. Mondrian - Troubleshooting

Error: Your current version of Flexmonster Pivot Table & Charts is not compatible with Flexmonster Accelerator. Please update the component to the minimum required version: X.XXX

This error means that the versions of Flexmonster Accelerator and Flexmonster Pivot are not compatible. To fix this error it is recommended to update both products to the latest available version. Contact our team (https://www.flexmonster.com/contact/) for more details.

Error: Your current version of Flexmonster Accelerator is not compatible with Flexmonster Pivot Table & Charts. Please update the Accelerator to the minimum required version: X.XXX

This error means that the versions of Flexmonster Accelerator and Flexmonster Pivot are not compatible. To fix this error it is recommended to update both products to the latest available version. Contact our team (https://www.flexmonster.com/contact/) for more details.

Error: java.lang.OutOfMemoryError: GC overhead limit exceeded

Try starting the Accelerator with the following: java -Xmx1024m -jar flexmonster-proxy-mondrian.jar where -Xmx<size> is the maximum Java heap size.

Error: java.net.UnknownHostException: <hostname>: <hostname>: Name or service not known

Some enterprise environments do not allocate DNS names to their devices – this results in an UnknownHostException, often after a lengthy DNS lookup attempt. Here are a couple of possible actions:

- The first suggestion is to ignore the message. The application is working fine and the message is only seen in the logs. Try opening your browser and navigating to http://127.0.0.1:50006/ or http://localhost:50006/. There should be a message “Flexmonster Accelerator for Pentaho Mondrian”.
- You can try adding the following record to the hosts file: 127.0.0.1 <hostname>.

4.11. Connecting to other data sources

This guide describes ways of connecting to the data sources that are not supported in Flexmonster out of the box (e.g., NoSQL databases or data warehouses such as Amazon Redshift (https://aws.amazon.com/redshift/)).

We suggest three approaches to connect the component to your data source:

- The easiest way: writing a server-side script that returns data in a JSON or CSV format – learn more here (#server-side-script).
- For .NET Core stack: using Flexmonster Data Server as a DLL, which allows implementing custom data fetching logic – learn more here (#fds-dll).
- For any stack: implementing a custom server that will retrieve data from your data source and pass it to Flexmonster – learn more here (#custom-server).

Get data from a data source with a server-side script

The steps below describe how to use a server-side script as a data provider for Flexmonster.

Step 1. Embed the component into your web page

If Flexmonster is not yet embedded, set up an empty component in your web page:

In pure JavaScript

Complete the Quick start (/doc/how-to-create-js-pivottable/) guide. Your code should look similar to the following example:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

In Angular

Complete the Integration with Angular (/doc/integration-with-angular/) guide. Your code should look similar to the following example:

```
<fm-pivot
 [toolbar]="true">
</fm-pivot>
```

In React

Complete the Integration with React (/doc/integration-with-react/) guide. Your code should look similar to the following example:

```
<FlexmonsterReact.Pivot
 toolbar={true}
/>
```

In Vue

Complete the Integration with Vue (/doc/integration-with-vue/) guide. Your code should look similar to the following example:

```
<Pivot
 ref="pivot"
 toolbar>
</Pivot>
```

Step 2. Prepare data from your data source

Write a server-side script that returns data from your data source in JSON or CSV format. The component will later use the URL to this server-side script. If you use Google BigQuery tables, see the guide for creating an application that returns data (https://cloud.google.com/bigquery/docs/quickstarts/quickstart-client-libraries).

Step 3. Enable cross-origin resource sharing (CORS)

By default, the browser prevents JavaScript from making requests across domain boundaries. CORS allows web applications to make cross-domain requests. Visit enable-cors.org (https://enable-cors.org/) to find information on how to setup CORS on different types of servers.

Step 4. Configure the report with your own data

Now it’s time to configure the pivot table on the web page. Let’s create a minimal report for this (replace type and filename with your specific values):

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true, report: {
        dataSource: {
            /* please pay attention to this property,
            *  Flexmonster needs information
            *  which data will be returned by the script;
            *  for CSV data specify type: "csv"*/
            type: "json",
            filename: "URL_to_your_server_side_script_returning_JSON_or_CSV"
        }
    }
});
```

Launch the web page from a browser — there you go! A pivot table is embedded into your project.

Get data from a data source with Flexmonster Data Server DLL

Flexmonster Data Server (https://www.flexmonster.com/doc/intro-to-flexmonster-data-server/) is a server-side tool that loads data from a data source and keeps it in RAM. When Flexmonster Pivot requests the data, the Data Server aggregates it and then passes to the component in a ready-to-show format.

Flexmonster Data Server DLL gives more flexibility in working with data. With the DLL, you can implement a custom data parser. This approach allows you to retrieve data from any data source in any format.

Moreover, the DLL provides ready-to-use methods that aggregate the data and send it to Flexmonster.

We have a set of detailed articles about using the Data Server as a DLL — check them out (https://www.flexmonster.com/doc/getting-started-with-data-server-dll/).

Get data from a data source with a custom server

Another option to visualize data from any data source is creating a custom server. The server should be responsible for fetching the data from a data source and processing it. Then it should pass the data to the component.

Flexmonster can handle data from any server with the custom data source API (https://www.flexmonster.com/doc/introduction-to-custom-data-source-api/). It is our custom protocol developed by the Flexmonster team. It allows retrieving already aggregated data from a server to Flexmonster Pivot.

You should implement this protocol on your server to send the data to the component. To learn how to implement the custom data source API on your server, refer to our step-by-step guides (https://www.flexmonster.com/doc/introduction-to-custom-data-source-api/).

5. Security

5.1. Security in Flexmonster

Security questions are always very important both for Flexmonster and for our customers. We place a lot of emphasis on data protection and apply various modern data protection techniques at Flexmonster.

There are two security aspects which are particularly significant for our customers:

- The data transfer process
- Data access management

The main objective of this section is to explain how to **handle all security requirements** while working with Flexmonster. More details are available in the following section:

- Security aspects of connecting to an OLAP cube (/doc/ways-of-connecting-to-an-olap-cube/)

5.2. Security aspects of connecting to an OLAP cube

5.2.1. Ways of connecting to an OLAP cube

Working with an OLAP cube requires proper data access management in order to provide a necessary level of security. There are two ways to connect to an OLAP cube: **XMLA protocol** and **Flexmonster Accelerator**.

XMLA protocol security

XMLA is a widely-known standard for data access. It supports roles – the main mechanism of access rights from SQL Server Analysis Services. Flexmonster Pivot is capable of getting the data via XMLA and roles are supported as well. For more instructions refer to the tutorial on using roles from Analysis Services (https://www.flexmonster.com/doc/configuring-authentication-process/#!roles).

Flexmonster Accelerator security

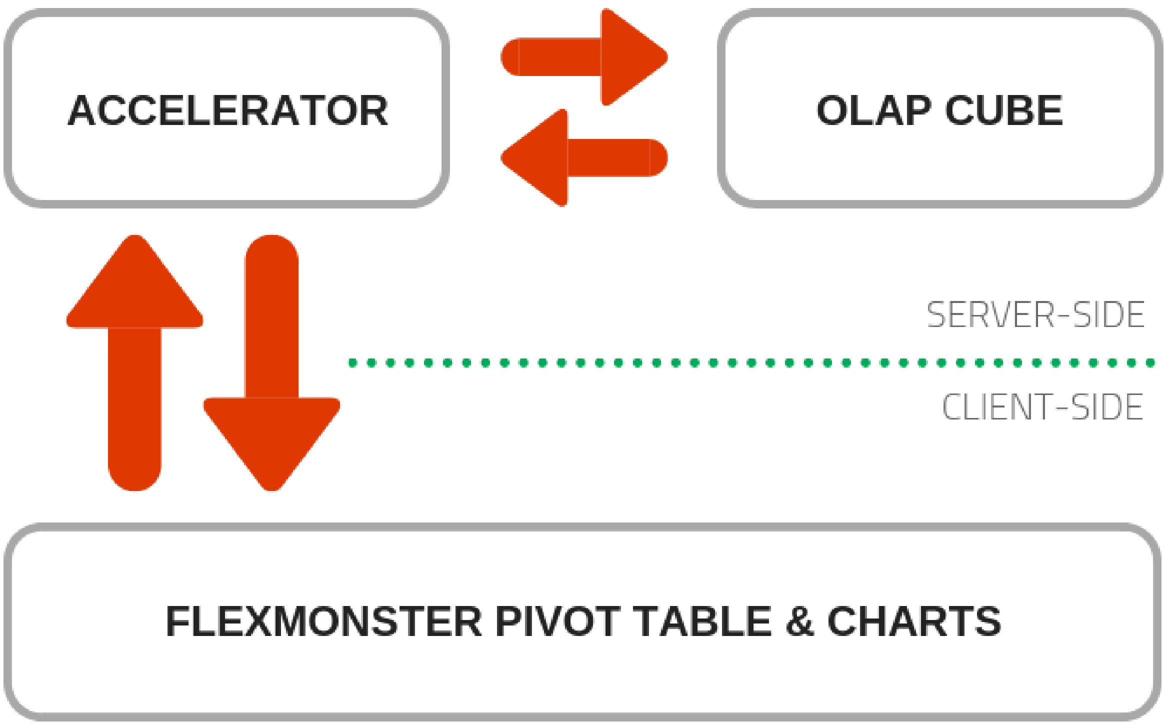
To provide more power for security configuration, our team developed a server-side proxy for connecting to the cubes. This proxy is called Flexmonster Accelerator and it has much more flexibility allowing to tune in the security properly. The Accelerator ships together with Flexmonster Pivot.

If your company needs more advanced data access management than roles, it is recommended to choose Flexmonster Accelerator for getting the data. All security aspects of this approach are covered in the sections below:

- The data transfer process (https://www.flexmonster.com/doc/data-transfer-accelerator/)
- Data security (https://www.flexmonster.com/doc/data-security-accelerator/)
- Data access management (https://www.flexmonster.com/doc/data-access-accelerator/)

5.2.2. The data transfer process

One of the most popular questions we get is **how is the data from the OLAP cube transferred** to Flexmonster Pivot? Flexmonster Accelerator serves as an additional server-side layer that helps to restrict external access to the database. When connecting to the data source inside the pivot table, the URL to the Accelerator is used instead of the SSAS server URL. Flexmonster Pivot sends the requests to the Accelerator, then Flexmonster Accelerator communicates with the SSAS server and gets the necessary data. This data is then sent back to the client from the Accelerator. The flowchart below depicts the process:



Data transfer security

To ensure **server-side data security**, the Accelerator doesn’t accept requests from any other web applications, only from Flexmonster Pivot. It is not possible to send an HTTP request directly to Flexmonster Accelerator without using Flexmonster Pivot. The Accelerator also doesn’t accept a response/request that was tampered with during the communication process. Each response/request contains a checksum for the package to ensure that it was not changed.

There is a requirement to open an additional port for the Accelerator on the server. This requirement is a strong restriction imposed by the browser’s security, not by Flexmonster.

Due to thesame-origin (https://en.wikipedia.org/wiki/Same-origin_policy)p (https://en.wikipedia.org/wiki/Same-origin_policy)olicy (https://en.wikipedia.org/wiki/Same-origin_policy), the browser only allows requests that come from the same origin. Cross-origin resource sharing (CORS) allows web applications to make cross-domain requests. Since the additional port for the Accelerator is opened, CORS must be enabled. Visit enable-cors.org (https://enable-cors.org/) to find out how to set up CORS on different types of servers.

Thus, it is necessary to use an extra port and enable CORS, and there is no workaround. Otherwise, the clients' browsers will not permit communication with the server.

5.2.3. Data security

This section describes data security concepts related to connecting to an OLAP cube.

Protecting the OLAP cube

As long as only the Accelerator needs to communicate with the client, it is highly recommended to **restrict any external access to the OLAP cube**. In that case, access to the cube will be restricted to only the local server. This increases security and protects against external threats including:

- Password cracking
- Unauthorized external access to the OLAP cube
- Data theft

If the SSAS server and the Accelerator are located on different servers, it is necessary to open a port on the SSAS server for the Accelerator.

Client-side data security

To ensure **client-side data security**, it's impossible to read data from the server without using Flexmonster Pivot. Each response/request contains a checksum for the packets. If this checksum is invalid, an error message is shown instead of the data.

HTTPS configuration

Even with the additional security between the Accelerator and Flexmonster Pivot, using the HTTP protocol is unsafe as it does not encrypt the data. HTTPS encrypts the data and protects it from inspection upon interception. **Flexmonster Accelerator supports HTTPS** and it is highly recommended that you use it instead of HTTP. For more details refer to the configuration tutorial (/doc/configuring-secure-https-connection/).

5.2.4. Data access management

It is often necessary to limit access to the data based on user roles. Flexmonster provides several options to **manage data access** levels with Flexmonster Accelerator. Below is a list of all available security models, each with a detailed tutorial.

- Configuring access rights by defining roles in SSAS (/doc/configuring-authentication-process/#!roles)
- Using Windows authentication (https://www.flexmonster.com/doc/configuring-authentication-process/#!windows-auth-accelerator)
- Using CustomData to make row filters dynamic (/question/cube-security-using-customdata-and-ssas/)

Custom authorization when using the Accelerator as a DLL

One of the installation options for the Accelerator is integrating it into the back end as a separate ASP.NET controller. This eliminates the need to configure firewall settings and simplifies the updating process. Using the Accelerator as a DLL also lets you use a custom authorization system, allowing you to manage security in any way you want. For more instructions refer to the custom authorization tutorial (/doc/configuring-authentication-process/#!custom-authorization).

6. Configuring report

6.1. What is a report

A report is a definition of what data should be shown in the component and how it should be visualized.

Reports are written in JSON. XML reports are also supported for backward compatibility.

Report objects were used to connect to JSON (/doc/json-data-source/), CSV (/doc/csv-data-source/), SQL databases (/doc/connect-to-relational-database/), MongoDB (/doc/mongodb-connector/), Microsoft Analysis Services (/doc/connecting-to-microsoft-analysis-services/), Pentaho Mondrian (/doc/connecting-to-pentaho-mondrian/), Elasticsearch (/doc/connecting-to-elasticsearch/), and custom data source API (/doc/introduction-to-custom-data-source-api/) in previous articles of the documentation.

This set of articles focuses on how to configure properties in a report.

A report object has many properties – all possible aspects of pivot table and pivot chart configuration can be set via a report object. The component supports saving and loading reports.

Report configuration

Report properties are logically divided into the parts. Follow the tutorials to configure each part:

- Data source (/doc/data-source/)
- Slice (/doc/slice/)
- Options (/doc/options/)
- Number formatting (/doc/number-formatting/)
- Conditional formatting (/doc/conditional-formatting/)
- Localization (/doc/localizing-component/)

If the data contains date and time values, check out the date and time formatting tutorial (/doc/date-and-time-formatting/).

Our Examples page contains a list of examples that demonstrate different ways of working with a report. Check them out (https://www.flexmonster.com/examples/#!configuring-report).

The simplest report

The simplest report consists of only the data source definition.

Let’s see how the simplest report that defines a connection to a static CSV file looks like in JSON:

```
{
 dataSource: {
  filename: "data.csv"
 }
}
```

This is how the above report is set in new Flexmonster():

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            filename: "data.csv"
        }
    }
});
```

Open on JSFiddle (https://jsfiddle.net/flexmonster/z235ddft/).

How to display non-English characters correctly

If you use an alphabet with special characters, it is necessary to set UTF-8 encoding. There are two ways to do this:

1. Specify the encoding for your HTML page to UTF-8 using the content-type HTTP header or the corresponding meta tag:

   ```
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   ```

   Content from the database or static report files must also be encoded as UTF-8.
2. If you are not able to change the header of your HTML file, you can embed Flexmonster in a separate JS file and specify UTF-8 encoding when referencing it:

   ```
   <script src="yourfile.js" charset="UTF-8"></script>
   ```

6.2. Data source

The data source is a required part of the report object. Flexmonster supports data from OLAP data sources, SQL databases, CSV and JSON static files, and inline JSON data. Each data source requires specific properties to be set inside the dataSource section of the report object. To see examples of connecting to different data sources, visit the Examples page (https://www.flexmonster.com/examples/#!data-source).

Read more in the following sections:

- JSON data sources (#json)
- CSV data sources (#csv)
- SQL databases (#sql)

- MongoDB databases (#mongo)
- The custom data source API (#custom-data-source-api)
- Data from OLAP data sources (#olap)
    - Microsoft Analysis Services (#msas)
    - Mondrian (#mondrian)
- Elasticsearch (#elasticsearch)
- Change the data source using the Toolbar (#toolbar)
- Data source via API (#api)

JSON data sources

JSON data source can be:

- Files from the local file system.
- A remote static file.
- Data generated by a server-side script.
- Inline JSON.

Here is a list of dataSource properties used to connect to JSON data sources:

- browseForFile – Boolean. Defines whether you want to load the file from the local file system (true) or not (false). *Default value: false.*
- data (from v2.2) – JSON. The inline JSON data.
- type – String. The type of data source. In this case, it is "json". You do not need to explicitly define this property when reading static JSON files with a “.json” extension.
- filename – String. The URL to the file or to the server-side script which generates data.
- mapping (optional) – Mapping Object (https://www.flexmonster.com/api/mapping-object/) | String. Allows defining field data types, captions, and multilevel hierarchies, grouping fields under separate dimensions, and setting other view configurations of hierarchies. It can be either the inline Mapping Object (https://www.flexmonster.com/api/mapping-object/) or the URL to a JSON file with mapping. See an example on JSFiddle (https://jsfiddle.net/flexmonster/5oxfubmc/).
- requestHeaders (optional) – Object. This object allows you to add custom request headers. The object consists of "key": "value" pairs, where "key" is a header name and "value" is its value. Check out a live sample on JSFiddle (https://jsfiddle.net/flexmonster/7z40n1r8/). Important note: requestHeaders is not saved when obtaining the report via save() and getReport() API calls.
- useStreamLoader (optional) – Boolean. Optimizes the large file processing using the stream loader. When set to true, the stream loader is enabled. Available only when loading files via URL. See an example on JSFiddle (https://jsfiddle.net/flexmonster/qd6h4tsv/). Default value: false.
- withCredentials (optional) – Boolean. It indicates whether cross-site Access-Control requests should be made using credentials such as authorization headers (true) or not (false). For more details refer to MDN web docs (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/withCredentials).
  Setting the withCredentials flag to true is recommended when using Windows authentication and other types of server authentications. When set to false, the browser does not request credentials, as well as does not include them into outgoing requests. *Default value: false.*

Here is an example of JSON dataset using a file from the local file system:

```
{
    dataSource: {
        /* Path to the local JSON file */
        filename: "data.json"
    }
}
```

If the data is generated by a server-side script, type must be defined explicitly:

```
{
    dataSource: {
        type: "json",
        filename: "script_which_returns_json_data"
    }
}
```

Inline JSON:

```
{
    dataSource: {
        /* jsonData variable contains JSON data */
        data: jsonData
    }
}
```

CSV data sources

CSV data source can be:

- A file from the local file system.
- A remote static file.
- Data generated by a server-side script.

Here is a list of dataSource properties used to connect to CSV data sources:

- browseForFile – Boolean. Defines whether you want to load a file from the local file system (true) or not (false). *Default value: false*.
- type – String. The type of data source. In this case, it is "csv". You do not need to explicitly define this property when reading static CSV files with a .csv extension.
- fieldSeparator (optional) – String. Defines the specific fields separator to split each CSV row. There is no need to define it if the CSV fields are separated by , or ;. This property is required only if another character separates fields.
  For example, if you use TSV, where a tab character is used to separate fields, then the fieldSeparator parameter should be set to "\t".
  Alternatively, you can specify the field separator in the CSV file’s first row using the sep prefix. Supported prefix formats are the following: sep=fieldSep and "sep=fieldSep". For example:

  ```
  sep=|
  Country|Color|Price
  Canada|blue|1000
  ```

- thousandSeparator (optional) – String. Allows importing CSV data with commas used to separate groups of digits in numbers (e.g., 1,000 for one thousand). To load such data, set thousandSeparator to ",".
- filename – String. The URL to the file or to the server-side script which generates data.
- ignoreQuotedLineBreaks (from v2.1) – Boolean. Indicates whether line breaks in quotes should be ignored (true) or not (false). When set to true, CSV parsing is faster. Set it to false only if your data source has important line breaks in quotes. Note that this might slow down CSV parsing a little bit. *Default value: true*.

- mapping (optional) – Mapping Object (https://www.flexmonster.com/api/mapping-object/) | String. Allows defining field data types, captions, and multilevel hierarchies, grouping fields under separate dimensions, and setting other view configurations of hierarchies. It can be either the inline Mapping Object (https://www.flexmonster.com/api/mapping-object/) or the URL to a JSON file with mapping. See an example on JSFiddle (https://jsfiddle.net/flexmonster/5oxfubmc/).
- requestHeaders (optional) – Object. This object allows you to add custom request headers. This object consists of "key": "value" pairs, where "key" is a header name and "value" is its value. Check out a live sample on JSFiddle (https://jsfiddle.net/flexmonster/7z40n1r8/). Important note: requestHeaders is not saved when obtaining the report via save() and getReport() API calls.
- withCredentials (optional) – Boolean. It indicates whether cross-site Access-Control requests should be made using credentials such as authorization headers (true) or not (false). For more details refer to MDN web docs (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/withCredentials). Setting the withCredentials flag to true is recommended when using Windows authentication and other types of server authentications. When set to false, the browser does not request credentials, as well as does not include them into outgoing requests. *Default value: false.*

In the following example data is taken from a CSV file where the colon character (:) is used to separate fields within the row. Line breaks in quotes are not ignored:

```
{
    dataSource: {
        /* URL or local path to a CSV file */
        filename: 'colon-data.csv',
        fieldSeparator: ':',
        ignoreQuotedLineBreaks: false
    }
}
```

If the data is generated by a server-side script, type must be defined explicitly:

```
{
    dataSource: {
        type: "csv",
        filename: "script_which_returns_csv_data"
    }
}
```

SQL databases

Here is a list of dataSource properties used to connect to SQL databases using the custom data source API:

- type – String. The type of data source. In this case, it is "api".
- url – String. The path to the API endpoints.
- index – String. The dataset identifier.
- mapping (optional) – Mapping Object (https://www.flexmonster.com/api/mapping-object/) | String. Allows defining field data types, captions, and multilevel hierarchies, grouping fields under separate dimensions, and setting other view configurations of hierarchies. It can be either the inline Mapping Object (https://www.flexmonster.com/api/mapping-object/) or the URL to a JSON file with mapping. See an example on JSFiddle (https://jsfiddle.net/flexmonster/5oxfubmc/).
- requestHeaders (optional) – Object. This object allows you to add custom request headers. This object

consists of "key": "value" pairs, where "key" is a header name and "value" is its value. Check out a live sample on JSFiddle (https://jsfiddle.net/flexmonster/7z40n1r8/). Important note: requestHeaders is not saved when obtaining the report via save() and getReport() API calls.
- withCredentials (optional) – Boolean. It indicates whether cross-site Access-Control requests should be made using credentials such as authorization headers (true) or not (false). For more details refer to MDN web docs (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/withCredentials). Setting the withCredentials flag to true is recommended when using Windows authentication and other types of server authentications. When set to false, the browser does not request credentials, as well as does not include them into outgoing requests. *Default value: false.*

Here is an example of how a connection to a relational database is represented in a dataSource object:

```
{
    dataSource: {
        type: "api",
        url: "https://olap.flexmonster.com:9202/api/cube",
        index: "fm-product-sales"
    }
}
```

Read more about connecting to SQL databases with the custom data source API here (/doc/connect-to-relational-database/).

MongoDB databases

Here is a list of dataSource properties used to connect to a MongoDB database:

- type – String. The type of data source. In this case, it is "api".
- url – String. The path to the API endpoints.
- index – String. The dataset identifier.
- mapping (optional) – Mapping Object (https://www.flexmonster.com/api/mapping-object/) **|** String. Allows defining field data types, captions, and multilevel hierarchies, grouping fields under separate dimensions, and setting other view configurations of hierarchies. It can be either the inline Mapping Object (https://www.flexmonster.com/api/mapping-object/) or the URL to a JSON file with mapping. See an example on JSFiddle (https://jsfiddle.net/flexmonster/5oxfubmc/).
- requestHeaders (optional) – Object. This object allows you to add custom request headers. This object consists of "key": "value" pairs, where "key" is a header name and "value" is its value. Check out a live sample on JSFiddle (https://jsfiddle.net/flexmonster/7z40n1r8/). Important note: requestHeaders is not saved when obtaining the report via save() and getReport() API calls.
- withCredentials (optional) – Boolean. It indicates whether cross-site Access-Control requests should be made using credentials such as authorization headers (true) or not (false). For more details refer to MDN web docs (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/withCredentials). Setting the withCredentials flag to true is recommended when using Windows authentication and other types of server authentications. When set to false, the browser does not request credentials, as well as does not include them into outgoing requests. *Default value: false.*

Here is an example of how a connection to a MongoDB database is represented in a dataSource object:

```
{
    dataSource: {
        type: "api",
        url: "https://olap.flexmonster.com:9204/mongo",
        index: "fm-product-sales"
    }
}
```

Check out a live example on JSFiddle (https://jsfiddle.net/flexmonster/hmkq852s/).

The custom data source API

Here is a list of dataSource properties used to connect to the custom data source API:

- type – String. The type of data source. In this case, it is "api".
- url – String. The path to the API endpoints.
- index – String. The dataset identifier.
- mapping (optional) – Mapping Object (https://www.flexmonster.com/api/mapping-object/) | String. Allows defining field data types, captions, and multilevel hierarchies, grouping fields under separate dimensions, and setting other view configurations of hierarchies. It can be either the inline Mapping Object (https://www.flexmonster.com/api/mapping-object/) or the URL to a JSON file with mapping. See an example on JSFiddle (https://jsfiddle.net/flexmonster/5oxfubmc/).
- requestHeaders (optional) – Object. This object allows you to add custom request headers. This object consists of "key": "value" pairs, where "key" is a header name and "value" is its value. Check out a live sample on JSFiddle (https://jsfiddle.net/flexmonster/7z40n1r8/). Important note: requestHeaders is not saved when obtaining the report via save() and getReport() API calls.
- singleEndpoint – Boolean. When set to true, all custom data source API requests are sent to a single endpoint specified in the url property. *Default value: false*.
- withCredentials (optional) – Boolean. It indicates whether cross-site Access-Control requests should be made using credentials such as authorization headers (true) or not (false). For more details refer to MDN web docs (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/withCredentials). Setting the withCredentials flag to true is recommended when using Windows authentication and other types of server authentications. When set to false, the browser does not request credentials, as well as does not include them into outgoing requests. *Default value: false.*

Here is an example of how a connection to the custom data source API is represented in a dataSource object:

```
{
    dataSource: {
        type: "api",
        url: "https://olap.flexmonster.com:9202/api/cube",
        index: "fm-product-sales"
    }
}
```

Check it out on JSFiddle (https://jsfiddle.net/flexmonster/j1c6rd24/).

Data from OLAP data sources

OLAP data sources include Microsoft Analysis Services and Mondrian. There are two ways to connect to an OLAP cube using Flexmonster Pivot:

1. Via the XMLA protocol – an industry standard for data access in analytical systems.
2. Via Flexmonster Accelerator – a special server-side utility developed by Flexmonster.

Microsoft Analysis Services

Here is a list of dataSource properties used to connect to Microsoft Analysis Services:

- catalog – String. The data source catalog name.
- cube – String. The given catalog's cube's name.
- dataSourceInfo (optional) – String. The service info.
- type – String. The type of data source. In this case, it is "microsoft analysis services".
- proxyUrl – String. The path to the proxy URL. Both tabular and multidimensional model types are supported.
- binary (optional) – Boolean. A flag to use Flexmonster Accelerator instead of the XMLA protocol.
- effectiveUserName (optional) – String. Use when an end-user identity must be impersonated on the server. Specify the account in a domain\user format.
- localeIdentifier (optional) – Number. The Microsoft locale ID value for your language.
- mapping (optional) – Mapping Object (https://www.flexmonster.com/api/mapping-object/) | String. Allows defining field data types, captions, multilevel hierarchies, grouping fields under separate dimensions, and setting other view configurations of hierarchies. It can be either the inline Mapping Object (https://www.flexmonster.com/api/mapping-object/) or the URL to a JSON file with mapping. See an example on JSFiddle (https://jsfiddle.net/flexmonster/5oxfubmc/).
- roles (optional) – String. A comma-delimited list of predefined roles to connect to a server or a database using the permissions defined by that role. If this property is omitted, all roles are used and the effective permissions are the combination of all roles. For example, to combine "admin" and "manager" roles, set the roles property like so: roles: "admin,manager".
- subquery (optional) – String. The parameter to set a server-side filter to decrease the size of the response from the OLAP cube. For example, to show reports for only one specific year set the subquery like so: "subquery": "select {[Delivery Date].[Calendar].[Calendar Year].&[2008]} on columns from [Adventure Works]".
- requestHeaders (optional) – Object. This object allows you to add custom request headers. This object consists of "key": "value" pairs, where "key" is a header name and "value" is its value. Check out a live sample on JSFiddle (https://jsfiddle.net/flexmonster/7z40n1r8/). Important note: requestHeaders is not saved when obtaining the report via save() and getReport() API calls.
- withCredentials (optional) – Boolean. It indicates whether cross-site Access-Control requests should be made using credentials such as authorization headers (true) or not (false). For more details refer to MDN web docs (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/withCredentials). Setting the withCredentials flag to true is recommended when using Windows authentication and other types of server authentications. When set to false, the browser does not request credentials, as well as does not include them into outgoing requests. This property is available for both XMLA protocol and Flexmonster Accelerator. *Default value: false.*

Here is an example of how the connection to SSAS via XMLA is represented in dataSource:

```
{
    dataSource: {
        type: "microsoft analysis services",
        // URL to msmdpump.dll
        proxyUrl: "https://olap.flexmonster.com/olap/msmdpump.dll",
        catalog: "Adventure Works DW Standard Edition",
        cube: "Adventure Works",
        // Microsoft Locale ID Value for French
        localeIdentifier: 1036,
        // roles from SSAS
        roles: "admin,manager",
        subquery: "select {
                    [Delivery Date].[Calendar].[Calendar Year].&[2008]
                   } on columns from [Adventure Works]"
    }
}
```

Check it out on JSFiddle (https://jsfiddle.net/flexmonster/hcckvwx0/).

Here is an example of how the connection to SSAS via Flexmonster Accelerator is represented in dataSource:

```
{
    dataSource: {
        type: "microsoft analysis services",
        proxyUrl: "http://localhost:50005",
        catalog: "Adventure Works DW Standard Edition",
        cube: "Adventure Works",
        binary: true,
        // Microsoft Locale ID Value for French
        localeIdentifier: 1036,
        subquery: "select {
                    [Delivery Date].[Calendar].[Calendar Year].&[2008]
                   } on columns from [Adventure Works]"
    }
}
```

You can read all the details about the Accelerator here (/doc/getting-started-with-accelerator-ssas/).

Mondrian

Here is a list of dataSource properties used to connect to Mondrian:

- catalog – String. The data source catalog name.
- cube – String. The given catalog's cube's name.
- dataSourceInfo – String. The service info.
- mapping (optional) – Mapping Object (https://www.flexmonster.com/api/mapping-object/) | String. Allows setting captions of dimensions and measures. It can be either the inline Mapping Object (https://www.flexmonster.com/api/mapping-object/) or the URL to a JSON file with mapping. See an example on JSFiddle (https://jsfiddle.net/flexmonster/5oxfubmc/).
- type – String. The type of data source. In this case it is "mondrian".
- proxyUrl – String. The path to the proxy URL.
- binary (optional) – Boolean. A flag to use Flexmonster Accelerator instead of the XMLA protocol.
- roles (optional) – String. A comma-delimited list of predefined roles to connect to a server or a database using the permissions defined by that role. If this property is omitted, all roles are used and the effective permissions are the combination of all roles. For example, to combine "admin" and "manager" roles, set the roles property like so: roles: "admin,manager".
- requestHeaders (optional) – Object. This object allows you to add custom request headers. This object consists of "key": "value" pairs, where "key" is a header name and "value" is its value. Check out a live sample on JSFiddle (https://jsfiddle.net/flexmonster/7z40n1r8/). Important note: requestHeaders is not saved when obtaining the report via save() and getReport() API calls.

Here is an example of how the connection to Mondrian via XMLA is represented in dataSource:

```
{
    dataSource: {
        type: "mondrian",
        // URL to the XMLA provider
        proxyUrl: "http://localhost:8080/mondrian/xmla",
        dataSourceInfo: "MondrianFoodMart",
        catalog: "FoodMart",
        cube: "Sales"
   }
}
```

Here is an example of how the connection to Mondrian via Flexmonster Accelerator is represented in dataSource:

```
{
    dataSource: {
        type: "mondrian",
        proxyUrl: "localhost:50006",
        dataSourceInfo: "MondrianFoodMart",
        catalog: "FoodMart",
        cube: "Sales",
        binary: true,
        roles: "California manager" // Mondrian roles
    }
}
```

You can read all the details about the Accelerator here (/doc/getting-started-with-accelerator-mondrian/).

Elasticsearch

Here is a list of dataSource properties used to connect to Elasticsearch:

- type – String. The type of the data source. In this case it is "elasticsearch".
- subquery (optional) – Bool Query Object (https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-bool-query.html). The parameter to set a server-side filter to decrease the size of the response from the server.
- mapping (optional) – Mapping Object (https://www.flexmonster.com/api/mapping-object/) | String. Allows customizing hierarchies' captions, formats, time zones, control fields' visibility, and more. It can be either the inline Mapping Object (https://www.flexmonster.com/api/mapping-object/) or the URL to a JSON file with mapping. See an example on JSFiddle (https://jsfiddle.net/flexmonster/5oxfubmc/).
- node – String. The URL string for the connection.
- index – String. The name of the Elasticsearch index to connect to.
- requestHeaders (optional) – Object. This object allows you to add custom request headers. This object consists of "key": "value" pairs, where "key" is a header name and "value" is its value. Check out a live sample on JSFiddle (https://jsfiddle.net/flexmonster/7z40n1r8/). Important note: requestHeaders is not saved when obtaining the report via the save() and getReport() API calls.
- withCredentials (optional) – Boolean. It indicates whether cross-site Access-Control requests should be made using credentials such as authorization headers (true) or not (false). For more details refer to MDN web docs (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/withCredentials). Setting the withCredentials flag to true is recommended when using Windows authentication and other types of server authentications. When set to false, the browser does not request credentials, as well as does not include them into outgoing requests. *Default value: false.*

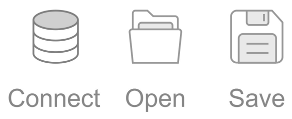
Here is an example of how a connection to Elasticsearch is represented in a dataSource object:

```
{
    dataSource: {
        type: "elasticsearch",
        node: "https://olap.flexmonster.com:9200",
        index: "fm-product-sales"
    }
}
```

Try the example on JSFiddle (https://jsfiddle.net/flexmonster/xfzws81u/).

Change the data source using the Toolbar

Use the Connect option to choose another data source or Open to load another report at runtime. Use Save to save the report with the current data source.



Data in the pivot table will be updated and saved within the report.

Data source via API

The API calls connectTo() (/api/connectto/), load() (/api/load/), and open() (/api/open/) are used to change the data source at runtime. The API call save() (/api/save/) is used to save the report.

6.3. Slice

A slice is a definition of the subset of data from your data source that will be shown in your report when you open it. By using slices in a report you can easily switch between different sets of values. To see different examples of slice configuration, visit our Examples page (https://www.flexmonster.com/examples/#!slice-configuration).

Read more in the following sections:

- Slice properties (#properties)
- Default slice (#default)
- Rows, columns, and measures (#values)
- Report filter (#report)
- Sorting (#sorting)
- Expands (#expands)
- Drills (#drills)
- Calculated values (#calculated)
- Change a slice using the Field List and controls on the grid (#change)
- Slice via API (#api)

Slice properties

You can define fields that go to rows, go to columns, go to measures, add filtering, sorting, report filtering, expands, and drills. Here is a list of all available properties for a slice:

- columns – Array of objects. A list of hierarchies selected in the report slice for columns. Each object can

have the following properties:
  - uniqueName – String. The hierarchy’s unique name.
  - caption (optional) – String. The hierarchy’s caption.
  - dimensionName (optional) – String. The dimension name.
  - filter (optional) – Filter Object (/api/filter-object/). Contains filtering information.
  - levelName (optional) – String. Used to specify the level of the hierarchy that is shown on the grid or on the chart.
  - showTotals (optional) — Boolean. Defines whether totals are shown or hidden for the hierarchy. When set to true, totals are shown. Only for the classic view.
    If showTotals is not specified, totals’ visibility is defined by the options.showTotals property. Learn more about options.showTotals (https://www.flexmonster.com/api/options-object/#showTotals).
    To show and hide totals via UI, use a context menu. Open it by right-clicking the hierarchy header.
  - sort (optional) – String. The sorting type for members ("asc", "desc" or "unsorted").
  - sortOrder (optional) – Array of strings. Set custom ordering for hierarchy members. Only for "csv" and "json" data source types. sortOrder can be specified like this: ["member_1", "member_2", etc.].
- drills (optional) – Object. Contains drill-down information in multilevel hierarchies.
  - drillAll (optional) – Boolean. Indicates whether all levels of all hierarchies in the slice will be drilled down (true) or drilled up (false).
  - columns (optional) – Array of objects. Used to save and restore drilled down columns.
  - rows (optional) – Array of objects. Used to save and restore drilled down rows.
- drillThrough (optional) – Array of strings. Allows pre-defining the slice for the drill-through view. Only for "csv", "json", and "api" data source types. drillThrough can be specified like this: ["Hierarchy name 1", "Hierarchy name 2", etc.] (see live demo (https://jsfiddle.net/flexmonster/ra6cbgoq/)).
- expands (optional) – Object. Stores information about the expansion of nodes.
  - expandAll (optional) – Boolean. Indicates whether all nodes in the data tree will be expanded (true) or collapsed (false) on the grid and on the charts.
  - columns (optional) – Array of objects. Used to save and restore expanded columns.
  - rows (optional) – Array of objects. Used to save and restore expanded rows.
- flatSort – Array of objects. Only for "json", "csv", and "api" data source types. It contains the list of objects defining the multicolumn sorting on the flat grid. Each object has the following properties:
  - uniqueName – String. The unique name of the hierarchy being sorted.
  - sort – String. The sorting type ("asc", "desc", or "undefined").

  Note: the hierarchies are sorted in the order they were specified (i.e., the first hierarchy is sorted first, and so on). Therefore, take the element’s order into account when defining the flat table multicolumn sorting.
  See the example on JSFiddle (https://jsfiddle.net/flexmonster/3u1o2mry/).
  To perform multicolumn sorting via UI, use Ctrl+click.
- measures – Array of objects. A list of the selected measures and those which have non-default properties. Each object has these properties:
  - uniqueName – String. The measure’s unique name.
  - active (optional) – Boolean. Indicates whether the measure will be selected for the report (true) or not (false). active: false can be useful if the measure has non-default properties, but should not be selected for the grid or the chart.
  - aggregation (optional) — String. The name of the aggregation that will be applied to the measure. To see the list of supported aggregation functions for each data source type, refer to Flexmonster’s technical specifications (/technical-specifications/#aggregations). If the measure is calculated, aggregation will be set to "none".
  - availableAggregations (optional) — Array of strings. Note that starting from version 2.8, the availableAggregations property is considered deprecated. Use the Mapping’s (/doc/mapping/#aggregations) aggregations property instead.
    availableAggregations represents the list of aggregation functions that can be applied to the current measure. If the measure is calculated, availableAggregations will be set to [].
  - caption (optional) – String. The measure’s caption.
  - formula (optional) – String. Represents the formula. Refers to the calculated measure (/doc/calculated-values/). It can contain the following operators: +, -, *, /. Other measures can be referenced using the measure’s unique name and the associated aggregation function, for

example sum("Price") or max("Order"). To see the list of supported aggregation functions for each data source type, refer to Flexmonster's technical specifications (https://www.flexmonster.com/technical-specifications/#aggregations).
    - individual (optional) – Boolean. Refers to the calculated measure. Defines whether the formula is calculated using raw values (true) or using aggregated values (false). Only for "json" and "csv" data source types. *Default value: false.*
    - calculateNaN (optional) – Boolean. Refers to the calculated measure. Defines whether the formula is calculated using NaN values (true) or using null values (false). *Default value: true*.
    - format (optional) – String. The name of the number formatting that will be applied to the measure. Measure values can be formatted according to the number formatting defined in the report. All available number formattings are stored in the formats array in the report. More information about the number formatting part of the report can be found in the number formatting (https://www.flexmonster.com/doc/number-formatting/) article.
    - grandTotalCaption (optional) – String. The measure's grand total caption.
- flatOrder – Array of strings. Defines the order of the hierarchies for the "flat" grid type. flatOrder can be specified like this: ["Hierarchy name 1", "Hierarchy name 2", etc.] (see live demo (https://jsfiddle.net/flexmonster/xj4py32w/)). Only for "json", "csv", and "api" data source types.
- memberProperties – Array of objects. Only for "microsoft analysis services" and "mondrian" data source types. Each object in the array has the following properties:
    - levelName – String. The hierarchy's unique name.
    - properties – Array of strings. Represents the properties to be shown on the grid. Other available member properties can be accessed through the context menu.
- reportFilters – Array of objects. A list of hierarchies selected in the report slice for report filters. Each object has the following properties:
    - uniqueName – String. The hierarchy's unique name.
    - caption (optional) – String. The hierarchy's caption.
    - dimensionName (optional) – String. The dimension name.
    - filter (optional) – Filter Object (/api/filter-object/). Contains filtering information.
    - levelName (optional) – String. Used to specify the level of the hierarchy that is shown on the grid or on the chart.
    - sort (optional) – String. The sorting type for members ("asc", "desc" or "unsorted").
    - sortOrder (optional) – Array of strings. Set custom ordering for hierarchy members. Only for "csv" and "json" data source types. sortOrder can be specified like this: ["member_1", "member_2", etc.].
- rows – Array of objects. A list of hierarchies selected in the report slice for rows. Each object can have the following properties:
    - uniqueName – String. The hierarchy's unique name.
    - caption (optional) – String. The hierarchy caption.
    - dimensionName (optional) – String. The dimension name.
    - filter (optional) – Filter Object (/api/filter-object/). Contains filtering information.
    - levelName (optional) – String. Used to specify the level of the hierarchy that is shown on the grid or on the chart.
    - showTotals (optional) — Boolean. Defines whether totals are shown or hidden for the hierarchy. When set to true, totals are shown. Only for the classic view.
      If showTotals is not specified, totals' visibility is defined by the options.showTotals property. Learn more about options.showTotals (https://www.flexmonster.com/api/options-object/#showTotals).
      To show and hide totals via UI, use a context menu. Open it by right-clicking the hierarchy header.
    - sort (optional) – String. The sorting type for members ("asc", "desc" or "unsorted").
    - sortOrder (optional) – Array of strings. Set custom ordering for hierarchy members. Only for "csv" and "json" data source types. sortOrder can be specified like this: ["member_1", "member_2", etc.].
- sorting (optional) – Object. Defines the sorting for numbers in a specific row and/or column in the pivot table.
    - column – Object. Defines the sorting for numbers in a specific column. This object has 3 properties:
        - tuple – Array of strings. Consists of unique names that identify the column in the table based on the column's data.

        - measure – Object. Identifies the measure on which sorting will be applied. Has the following properties:
            - uniqueName – String. The measure’s unique name.
            - aggregation (optional) – String. The measure’s aggregation type.
        - type – String. The sorting type ("asc" or "desc").
    - row – Object. Defines the sorting for numbers in a specific row. This object has 3 properties:
        - tuple – Array of strings. Consists of unique names that identify the row in the table based on the row’s data.
        - measure – Object. Identifies the measure on which sorting will be applied. Has the following properties:
            - uniqueName – String. The measure’s unique name.
            - aggregation (optional) – String. The measure’s aggregation type.
        - type – String. The sorting type ("asc" or "desc").

Default slice

If a slice was not defined, Flexmonster will select a default slice for the report, where the first hierarchy from the data goes to rows and the first measure goes to columns. The automatic default slice selection is available for JSON and CSV data sources (it is not available for OLAP). You can turn off the default slice by setting showDefaultSlice in options to false. For example, take a look at the JSON data below:

```
var jsonData =  [
    {
        "Category": "Accessories",
        "Price": 125,
        "Quantity": 100
    },
    {
        "Category": "Accessories",
        "Price": 74,
        "Quantity": 235
    },
    {
        "Category": "Bikes",
        "Price": 233,
        "Quantity": 184
    }
]
```

"Category" is the first hierarchy, so it goes to rows. "Price" is the first measure, so it goes to columns. For this dataset, the default slice will look like this:

```
{
    "dataSource": {
        "data": jsonData
    },
    "slice": {
        "rows": [
            {
                "uniqueName": "Category"
            }
        ],
        "columns": [
            {
                "uniqueName": "[Measures]"
            }
        ],
        "measures": [
            {
                "uniqueName": "Price"
            }
        ]
    }
}
```

See the same example with a default slice on JSFiddle (https://jsfiddle.net/flexmonster/22ospzoa/).

Rows, columns, and measures

Starting from version 2.304, a slice can be defined with only measures:

```
{
    "dataSource": {
        "filename": "https://www.flexmonster.com/download/data.csv"
    },
    "slice": {
        "measures": [
            {
                "uniqueName": "Price",
                "aggregation": "sum",
                "active": true
            }
        ]
    }
}
```

Open on JSFiddle (https://jsfiddle.net/flexmonster/xnfyLff0/).

"uniqueName": "[Measures]" allows you to define where the measures will be displayed (in rows or in columns). By default they go to columns. Here is an example of a slice with rows, columns, and measures:

```
{
    "dataSource": {
        "filename": "https://www.flexmonster.com/download/data.csv"
    },
    "slice": {
        "rows": [
            {
                "uniqueName": "Category",
                "filter": {
                    "members": [
                        "category.[cars]",
                        "category.[bikes]"
                    ]
                },
                "sort": "desc"
            }
        ],
        "columns": [
            {
                "uniqueName": "[Measures]"
            },
            {
              "uniqueName": "Color",
                "filter": {
                    "members": [
                        "color.[blue]"
                    ]
                }
             }
        ],
        "measures": [
            {
                "uniqueName": "Price",
                "aggregation": "sum",
                "active": true
            },
            {
                "uniqueName": "Discount",
                "aggregation": "min",
                "active": true
            }
        ]
    }
}
```

See the same example on JSFiddle (https://jsfiddle.net/flexmonster/a2mguf82/).

Report filter

A report filter allows you to display different data subsets in the report. In Flexmonster Pivot the reportFilters property is used for defining the report filters as follows:

```
{
    "dataSource": {
        "filename": "https://www.flexmonster.com/download/data.csv"
    },
    "slice": {
        "reportFilters": [
            {
                "uniqueName": "Color",
                "filter": {
                    "members": [
                        "color.[yellow]",
                        "color.[white]"
                    ]
                }
            }
        ],
        "rows": [
            {
                "uniqueName": "Category"
            }
        ],
        "columns": [
            {
                "uniqueName": "[Measures]"
            }
        ],
        "measures": [
            {
                "uniqueName": "Price"
            }
        ]
    }
}
```

See an example with a report filter on JSFiddle (https://jsfiddle.net/flexmonster/7jpbqjsh/).

Sorting

The sorting object defines the sorting for numbers in a specific row and/or column in the pivot table. Sorting is usually configured on the grid and then saved within the report. It looks like this:

```
"sorting": {
    "column": {
        "type": "desc",
        "tuple": [],
        "measure": "Price"
    }
}
```

Sorting for members of columns and rows is defined like this:

```
"rows": [
    {
        "uniqueName": "Category",
        "filter": {
            "members": [
                "category.[cars]",
                "category.[bikes]"
            ]
        },
        "sort": "desc"
    }
]
```

Expands

Information about expands and collapses is stored within the slice. When a customer performs one of these operations, all the changes can be saved within the report. Use the expandAll property to apply the operation to all levels of data detail at once. This is how an expands object looks:

```
"expands": {
    "expandAll": false,
    "rows": [
        {
            "tuple": [
                "category.[accessories]"
            ]
        },
        {
            "tuple": [
                "category.[cars]"
            ]
        }
    ]
}
```

Drills

The drills object is used to store information about drill-downs in multilevel hierarchies. Here is an example of a drills object:

```
"drills": {
    "drillAll": false,
    "rows": [
        {
            "tuple": [
                "category.[accessories]"
            ]
        },
        {
            "tuple": [
                "category.[accessories].[bike racks]"
            ]
        },
        {
            "tuple": [
                "category.[accessories].[bottles and cages]"
            ]
        }
    ]
}
```

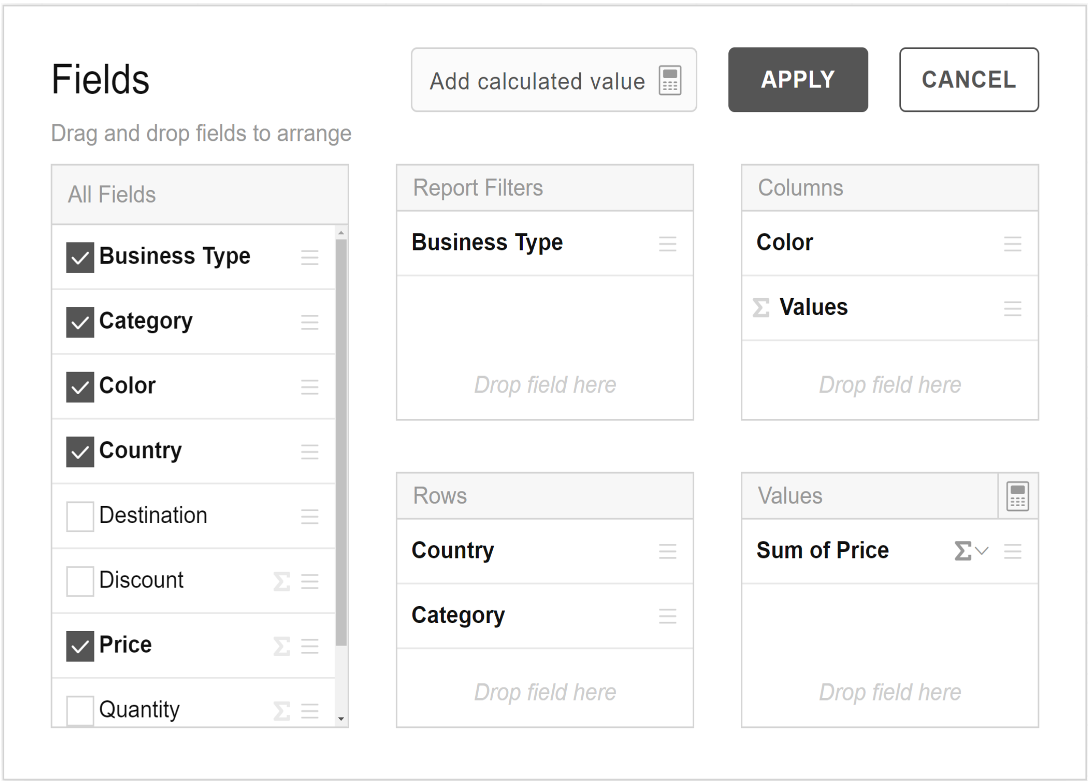
Calculated values

You can create as many calculated measures for one report as you need. When you save the report all the calculated measures will be saved as well and loaded when the report is retrieved. Note that you can add calculated measures only for reports that are based on a "csv", "json", or "api" data source type. Below is an example of a calculated measure:

```
"measures": [
    {
        "uniqueName": "Avg",
        "formula": "sum('Price') / count('Category') ",
        "caption": "Avg",
        "active": true
    }
]
```

Change a slice using the Field List and controls on the grid

Use the Field List to define report filters, rows, columns, and values at run time.



Sorting, filtering, drill-ups, drill-downs, and expand/collapse operations are available directly on the grid and on built-in pivot charts.

Slice via API

You can change the slice along with other report parts using the API call setReport() (/api/setreport/). To change only the slice use the runQuery() (/api/runquery/) call.

6.4. Options

Flexmonster’s appearance can be defined within the report. Options are used to specify the appearance and functionality available to customers. For example, you can show/hide features such as filters and sorting controls, define date patterns, or enable/disable the drill-through.

Options are defined in an options object inside the report. If no options are specified, Flexmonster will use the default options. This tutorial explains which options are available and how to use them.

Our default options are defined automatically, so you only need to define the options you want to modify. All options can be divided into three subgroups:

1. Grid options (#grid) – define the grid’s appearance and functionality.
2. Chart options (#chart) – define the chart’s appearance and functionality.
3. General options (#general-options) – define the options applicable to the whole component.

Read this (#set-options)section to learn how to set the options. The default options (#default) section contains an example of Flexmonster’s default options.

Visit our Examples page (https://www.flexmonster.com/examples/#!options) for live examples that demonstrate how to use different options.

Grid options

All grid options are in the grid section of the options object.

Available grid options

grid.type	String. The selected grid’s type. The following grid types are supported: "compact", "classic", and "flat". *Default value: "compact"*.
grid.title	String. The title of the grid. *Default value: ""*.
grid.showFilter	Boolean. Indicates whether column/row filter controls and report filter controls are visible on the grid (true) or not (false). *Default value: true*.
grid.showHeaders	Boolean. Indicates whether the spreadsheet headers are visible (true) or not (false). *Default value: true*.
grid.showTotals	String. Specifies where to show totals: in rows ("rows"), in columns ("columns"), in rows and columns ("on") or not at all ("off"). *Default value: "on"*.
grid.showGrandTotals	String. Specifies where to show grand totals: in rows ("rows"), in columns ("columns"), in rows and columns ("on") or not at all ("off"). *Default value: "on"*.
grid.grandTotalsPosition	String. Indicates whether the grand totals are displayed at the top of the table ("top") or at the bottom ("bottom"). Only available for the flat view. *Default value: "top"*.
grid.showHierarchies	Boolean. Specifies how to show drillable hierarchy cells on the grid: with a link on the right (true) or with an icon on the left (false). *Default value: true*.
grid.showHierarchyCaptions	Boolean. Indicates whether the hierarchy captions are visible (true) on the grid or not (false). *Default value: true*.
grid.drillthroughMaxRows	Number. Sets the maximum number of rows for the SSAS drill-through pop-up window. *Default value: 1000*.

grid.showReportFiltersArea	Boolean. Indicates whether the report’s filtering cells on the grid should be visible (true) or not (false). *Default value: true*.
grid.dragging	Boolean. Indicates whether the hierarchies on the grid can be dragged (true) or not (false). *Default value: true*.
grid.showAutoCalculationBar	Boolean. Indicates whether the autoCalculationBar feature (/user-interface/#autoCalculationBar) is turned on (true) or not (false). *Default value: true*.

Check out a live demo showing grid options

Chart options

All chart options are in the chart section of the options object.

Available chart options

chart.type	String. The selected chart type. The following chart types are supported: "column", "bar_h", "line", "scatter", "pie", "stacked_column", and "column_line". *Default value: "column"*.
chart.title	String. The title of the chart. *Default value: ""*.
chart.showFilter	Boolean. Indicates whether column/row filter controls and report filter controls are visible on the charts (true) or not (false). *Default value: true*.
chart.multipleMeasures	Boolean. Available from v1.9. Indicates whether to show multiple measures on charts. *Default value: false*.
chart.oneLevel	Boolean. In a drillable chart, defines whether the chart shows all nodes on the x-axis and the legend (false) or only the lowest expanded node on the x-axis and the legend (true). *Default value: false*.
chart.autoRange	Boolean. Indicates whether the range of values in the charts is selected automatically or not. *Default value: false*.
chart.reversedAxes	Boolean. Reverses the columns and rows when drawing charts. *Default value: false*.
chart.showLegend	Boolean. Indicates whether the legend for the charts is visible (true) or not (false). *Default value: true*.
chart.showLegendButton	Boolean. Indicates whether the button to show/hide the legend for the charts is visible. When set to false, the legend is visible, without a button to hide it. *Default value: false*.

chart.showDataLabels	Boolean. Setting this value to true allows showing labels in charts. If the value is false, the labels will be hidden. Use showAllLabels to configure the labels in a pie chart. *Default value: false*.
chart.showAllLabels	Boolean. Setting this value to true allows showing all the labels in a pie chart. If the value is false then only the labels that can be shown without overlapping will be shown. *Default value: false*.
chart.showMeasures	Boolean. Hides all dropdowns on the tops of the charts. This is useful if you want to show a simple chart without any controls or if you want to save space. When set to true, the dropdowns are visible. *Default value: true*.
chart.showOneMeasureSelection	Boolean. When set to true, the measures dropdown is always visible – regardless of the number of measures in it. If the value is set to false, the measures dropdown on charts will be hidden if there is only one measure in the list and visible if there are two or more measures. *Default value: true*.
chart.showWarning	Boolean. Indicates whether warnings are shown if the data is too big for charts. *Default value: true*.
chart.position	String. The positions of the charts in relation to the grid. It can be "bottom", "top", "left", or "right". *Default value: "bottom"*.
chart.activeMeasure	Object. Identifies the selected measure in the charts view.
chart.activeMeasure.uniqueName	String. The unique measure name. This measure must be present in the slice.measures (/api/slice-object/#measures) property.
chart.activeMeasure.aggregation	String. The measure aggregation type. This aggregation must be defined for the measure in the slice.measures.aggregation (/api/slice-object/#aggregation) property.
chart.pieDataIndex	String. The selected column member index. Learn more (/doc/flexmonster-pivot-charts/#pie-chart) about the pie chart structure. Indexes start from "0" (the first member of a column field). If pieDataIndex is set to a non-existing index, the default index will be used. *Default value: "0"*.
chart.axisShortNumberFormat	Boolean. Indicates whether axes labels on charts are displayed using a short number format like 10K, 10M, 10B, 10T (true) or not (false). *Default value: undefined (show short format if the max value > 10M)*.

Check out a live demo showing chart options

General options

All the general options are in the options object.

Available general options

viewType	String. The type of view to show: "grid" or "charts" or "grid_charts" (starting from v1.9). *Default value: "grid"*.
filter	Object. Filtering options:
filter.timezoneOffset	Number. The difference (in minutes) between UTC and the user's local time zone. Used to specify time zone for the dates in the date query filter (https://www.flexmonster.com/api/date-query-object/). *Default value: the user's local time*.
filter.weekOffset	Number. Sets the number of days to be added to the start of the week (Sunday). Used to adjust the first day of the week in the filter's calendar. *Default value: 1 (Monday is the first day of the week)*.
filter.dateFormat	String. The date pattern to use to format dates in the filter's date inputs. Has two possible values: "dd/MM/yyyy" and "MM/dd/yyyy". *Default value: "dd/MM/yyyy"*.
filter.liveSearch	Boolean. Indicates whether the search in the filter pop-up window is performed while the user types (true) or requires the Enter button to start searching (false). *Default value: true*.
configuratorActive	Boolean. Indicates whether the Field List is opened (true) or closed (false) after the component is initialized. *Default value: false*.
configuratorButton	Boolean. Indicates whether the Field List toggle button is visible (true) or not (false). *Default value: true*.
showAggregations	Boolean. Indicates whether the aggregation selection control is visible (true) or not (false) for measures on the Field List. *Default value: true*.
showCalculatedValuesButton	Boolean. Controls the visibility of "Add calculated value" in the Field List. *Default value: true*.
showEmptyValues	Boolean | String. Specifies where to show members with empty values on the grid and charts: in rows ("rows"), in columns ("columns"), in rows and columns (true), or not at all (false). Only for "csv" and "json" data source types. *Default value: false*.
grouping	Boolean. Indicates whether grouping is enabled. This feature allows customers to group chosen elements using a filter window. For example, if the customer has shops in different cities and wants to analyze sales information, it would be possible to combine several cities in one group by geography, by sales numbers, etc. Only available for "csv" and "json" data source types. *Default value: false*.
editing	Boolean. Indicates whether the editing feature is enabled (true) or disabled (false) on the drill-through pop-up window for CSV and JSON data sources. The user will

	be able to double-click the cell and enter a new value into it if the editing feature is enabled. *Default value: false*.
drillThrough	Boolean. Indicates whether the drill-through feature is enabled (true) or disabled (false). The user can drill through by double-clicking the cell with a value. *Default value: true*.
showDrillThroughConfigurator	Boolean. Indicates whether the Field List toggle button is visible in the drill-through view. Only for "csv", "json", "api", "microsoft analysis services" (for Flexmonster Accelerator), and "elasticsearch" data source types. *Default value: true*.
sorting	String. Indicates whether the sorting controls are visible in rows ("rows"), in columns ("columns"), in rows and columns ("on" or true) on the grid cells, or not visible at all ("off" or false). *Default value: "on"*.
readOnly	Boolean. When set to true, enables the read-only mode for the component. In the read-only mode, you cannot interact with the report via UI (e.g., the context menu, expands, and drills are disabled). In addition, configurations of the following options are ignored: showFilter (for both grid and charts), dragging, showMeasures, configuratorButton, sorting, and drillThrough. We also suggest hiding the Toolbar when using the read-only mode. *Default value: false*.
strictDataTypes	Boolean. When set to true, the component treats fields marked as a measure only as measures and does not allow using them as hierarchies. This allows increasing data analyzing speed. To mark a field as a measure, set its mapping.isMeasure (https://www.flexmonster.com/api/mapping-object/#isMeasure) property to true. Check out an example (https://jsfiddle.net/flexmonster/b9xy0j5u/). Only for the "json" data source type. *Default value: false*.
distinguishNullUndefinedEmpty	Boolean. Allows distinguishing null, undefined, and empty values. When set to false, the component treats null, undefined, and empty values as the same value. When set to true, there are three different values. *Default value: false*.
defaultDateType	String. Used to specify which data types should be applied to date fields by default ("date", "date string", "year/month/day", "year/quarter/month/day", or "datetime"). Only for "json" and "csv" data source types. *Default value: "date"*.
datePattern	String. Allows specifying how the component displays fields of the "date string" type. Only for "json", "csv", "api", and "elasticsearch" data source types. *Default pattern string: "dd/MM/yyyy"*. Learn more about date and time formatting (https://www.flexmonster.com/doc/date-and-time-formatting/).
dateTimePattern	String. Allows specifying how the component displays fields of the "datetime" type. Only for "json", "csv", and "api" data source types. *Default pattern string: "dd/MM/yyyy HH:mm:ss"*.

	Learn more about date and time formatting (https://www.flexmonster.com/doc/date-and-time-formatting/).
timePattern	String. Allows specifying how the component displays fields of the "time" type. Only for "json", "csv", and "api" data source types. *Default pattern string: "HH:mm:ss"*. Learn more about date and time formatting (https://www.flexmonster.com/doc/date-and-time-formatting/).
dateTimezoneOffset	Number. Allows setting the time zone for hierarchical dates (for JSON: "date", "year/month/day", and "year/quarter/month/day"; for CSV: "d+", "D+", and "D4+"). See an example (https://jsfiddle.net/flexmonster/r8dk7v63/).
saveAllFormats	Boolean. If there are more than 20 formats defined, only the formats that are used for "active: true" measures will be saved in the report. In order to save all the formats, set the saveAllFormats property to true. *Default value: false*.
showDefaultSlice	Boolean. Defines whether the component selects a default slice for a report with an empty slice (when nothing is set in rows, columns, report filters, and measures). If true, the first hierarchy from data goes to rows and the first measure goes to columns in the default slice. To avoid this default behavior, set this property to false. Only available for "csv" and "json" data source types. *Default value: true*.
useOlapFormatting	Boolean. Indicates whether the values from data source will be formatted according to the format defined in the cube (true) or not (false). *Default value: false*.
showMemberProperties	Boolean. Indicates whether the member properties for an OLAP data source are available in the component (true) or not (false). This feature is only for "microsoft analysis services" and "mondrian" data source types. *Default value: false*.
showEmptyData	Boolean. It defines the component’s behavior in case of an empty data source. Flexomontser handles the empty data in two ways: • when showEmptyData is set to true, the component will show an empty grid for an empty CSV/JSON file, a JSON file with an empty array, or an empty inline JSON array. For an empty CSV data source with the defined header, the component will show the slice with empty data cells. • when showEmptyData is set to false, the component will show a message “Data source is empty” and trigger a dataerror event for all mentioned types of the empty data source. See an example on JSFiddle (https://jsfiddle.net/flexmonster/4ku7Lscz/). *Default value: true.*
defaultHierarchySortName	String. The sorting type for hierarchies’ members ("asc", "desc", or "unsorted"). *Default value: "asc"*.
showOutdatedDataAlert	Boolean. Setting this value to true will show a warning to

	the user before automatic reloading of data from the cube. There will be no warnings if set to false. Only for Flexmonster Accelerator. *Default value: false*.
expandExecutionTimeout	Number. Allows specifying an execution timeout for the expandAllData() (https://www.flexmonster.com/api/expandalldata/) function. The timeout is set in milliseconds (i.e., 9000 equals 9 seconds). The minimum timeout value is 9000. Note that a large execution timeout can cause an unresponsive script alert. *Default value: 9000*.
showAggregationLabels	Boolean. Indicates whether aggregation labels like "Total Sum of", "Sum of", etc. are shown in the column/row titles. *Default value: true*.
showAllFieldsDrillThrough	Boolean. Indicates whether prefiltering the drill-through view columns is enabled (false) or if the drill-through view displays all the available columns (true) when using SSAS with Flexmonster Accelerator. *Default value: false*.
liveFiltering	Boolean. Indicates whether the live filtering for hierarchies’ members is enabled (true) or disabled (false). *Default value: false*.
showFieldListSearch	Boolean. Indicates whether the search bar in the Field List is shown (true) or hidden (false). When the search bar is hidden, it will be shown only when the number of the hierarchies exceeds 50 (or 40 for the flat form). *Default value: false*.
useCaptionsInCalculatedValueEditor	Boolean. By default, Flexmonster uses field unique names in the calculated value editor. If useCaptionsInCalculatedValueEditor is set to true, the component will use field captions instead of unique names. *Default value: false*.
validateFormulas	Boolean. Indicates whether validation is performed for calculated values (true) or not (false). If validation is turned on and the report contains an invalid formula, the “Wrong formula format” alert message is shown. To turn off the “Wrong formula format” alert message, set the validateFormulas property to false. *Default value: true*.
caseSensitiveMembers	Boolean. Indicates whether the hierarchies’ members are case-sensitive (true) or not (false). *Default value: false*.
simplifyFieldListFolders	Boolean. Indicates whether the folders containing one field should show this field in the root (true) or not (false). Only for Elasticsearch and SSAS data sources. *Default value: false*.
validateReportFiles	Boolean. Indicates whether validation of report files is turned on (true) or turned off (false). Setting this value to false allows loading report files in the old format without an error message. Should be used in global options. *Default value: true*.
fieldListPosition	String. Indicates whether the Field List is always shown on the right ("right") or in the pop-up window (undefined). *Default value: undefined*.
allowBrowsersCache	Boolean. Indicates whether browsers are allowed to cache the data (true) or not (false). When

allowBrowsersCache is set to false, caching is not allowed, and Flexmonster appends a unique id to the data source's URL to ensure that the URL stays unique. The unique URL prevents the browser from loading the data from its cache, so the data is updated every time it is requested. Setting allowBrowsersCache to true allows caching, and the URL remains unmodified (check out an example (https://jsfiddle.net/flexmonster/1c73vtup/)). *Default value: false.*

Check out a live demo showing options

How to set options

Options can be set in several ways:

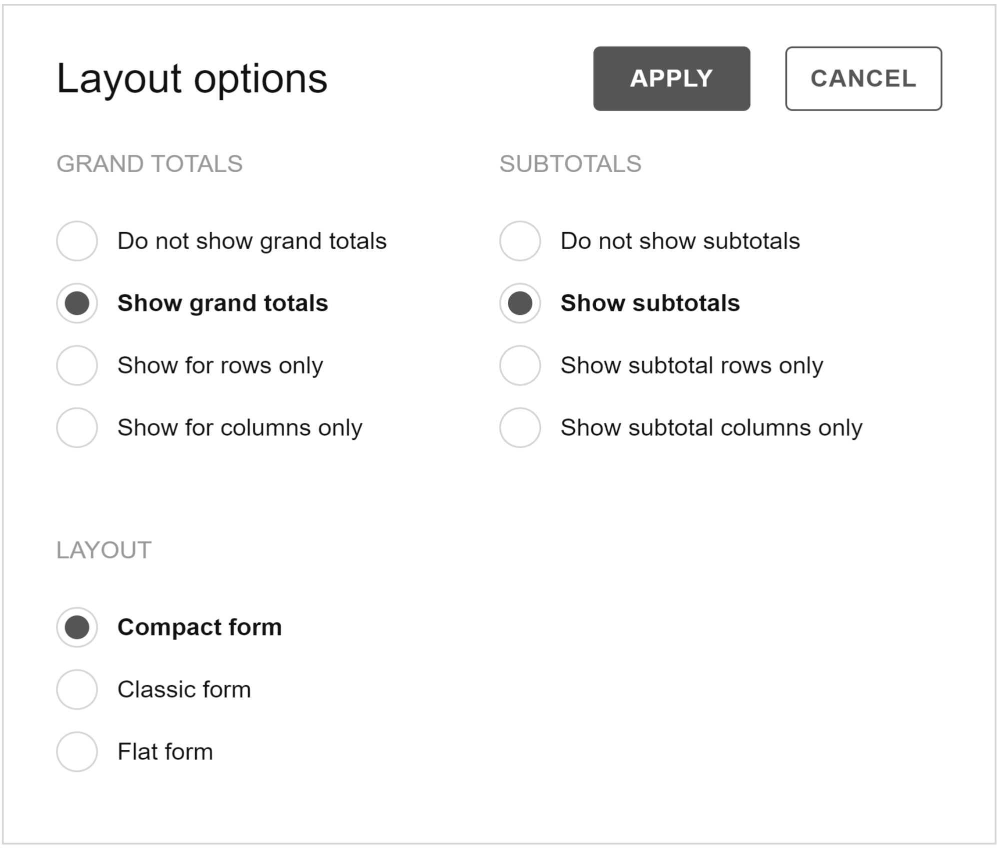
1. The most convenient way to set options is to specify them in the report. Here is how a grid title can be added:

   ```
   {
       "dataSource": {
           "filename": "https://www.flexmonster.com/download/data.csv"
       },
       "options": {
           "grid": {
               "title": "Results"
           }
       }
   }
   ```

   Check out a live sample on JSFiddle (https://jsfiddle.net/flexmonster/2rn0bxgo/).
2. You can change options along with other report parts using API call setReport() (/api/setreport/). Options can also be changed separately via setOptions() (/api/setoptions/). To see the current options use getOptions() (/api/getoptions/). For example, to set the chart title:

   ```
   flexmonster.setOptions({ chart: { title: "Chart One" } });
   flexmonster.refresh();
   ```

3. Some options can be changed using the Toolbar. Use Options in the Toolbar to change grand totals, subtotals, and the table layout at runtime.


Layout options

GRAND TOTALS

- Do not show grand totals
- [✓ **Show grand totals**]
- Show for rows only
- Show for columns only

SUBTOTALS

- Do not show subtotals
- [✓ **Show subtotals**]
- Show subtotal rows only
- Show subtotal columns only

LAYOUT

- [✓ **Compact form**]
- Classic form
- Flat form

Default options

If an options object is not defined in the report, Flexmonster Pivot will use global options if they are defined, or defaults from the component. These options can be overridden in the report.

Example of default options

```
{
   "dataSource": {
        "filename": "https://www.flexmonster.com/download/data.csv"
    },
    "options": {
        "viewType": "grid",
        "grid": {
            "type": "compact",
            "title": "",
            "showFilter": true,
            "showHeaders": true,
            "showTotals": "on",
            "showGrandTotals": "on",
            "grandTotalsPosition": "top",
            "showExtraTotalLabels": false,
            "showHierarchies": true,
            "showHierarchyCaptions": true,
            "drillthroughMaxRows": 1000,
            "showReportFiltersArea": true,
            "dragging": true,
            "showAutoCalculationBar": true,
            "showEmptyValues": false
        },
        "chart": {
            "type": "column",
            "title": "",
            "showFilter": true,
            "multipleMeasures": false,
            "oneLevel": false,
            "autoRange": false,
            "reversedAxes": false,
            "showLegend": true,
            "showLegendButton": false,
            "showDataLabels": false,
            "showAllLabels": false,
            "showMeasures": true,
            "showOneMeasureSelection": true,
            "showWarning": true,
            "activeMeasure": {}
        },
        "filter": {
            "weekOffset": 1,
            "dateFormat": "dd/MM/yyyy",
            "liveSearch": true
        },
        "configuratorActive": false,
        "configuratorButton": true,
        "showAggregations": true,
        "showCalculatedValuesButton": true,
        "grouping": false,
        "editing": false,
        "drillThrough": true,
        "showDrillThroughConfigurator": true,
        "sorting": "on",
        "defaultDateType": "date",
        "strictDataTypes": false,
        "datePattern": "dd/MM/yyyy",
        "dateTimePattern": "dd/MM/yyyy HH:mm:ss",
        "saveAllFormats": false,
        "showDefaultSlice": true,
        "useOlapFormatting": false,
        "showMemberProperties": false,
        "showEmptyData": true,
        "defaultHierarchySortName": "asc",
        "showOutdatedDataAlert": false,
        "showAggregationLabels": true,
        "sortAlphabetically": [],
        "showAllFieldsDrillThrough": false,
        "liveFiltering": false,
        "showFieldListSearch": false,
        "validateFormulas": true,
        "caseSensitiveMembers": false,
        "simplifyFieldListFolders": false,
        "validateReportFiles": true,
        "fieldListPosition": undefined,
        "allowBrowsersCache": false
    }
}
```

What's next?

You may be interested in the following articles:

- How to define a format for date and time (/doc/date-and-time-formatting/)
- How to set specific options common for all reports (/doc/global-object/)
- How to customize the Toolbar (/doc/customizing-toolbar/)
- How to add localization (/doc/localizing-component/)
- How to customize appearance with CSS (/doc/customizing-appearance/)
- How to configure extended grid customization (/doc/extended-grid-customization/)

6.5. Mapping

Mapping is the process of defining how fields contained in the data source are treated and presented within the component. For mapping in Flexmonster, you can use the Mapping object, which is a property of the Data Source (https://www.flexmonster.com/doc/data-source/).

The Mapping object is available for all data sources but with some differences.

For JSON, CSV, and the custom data source API, it's possible to define field data types and captions, group fields under separate dimensions, create multilevel hierarchies, and more. For SSAS and Mondrian data sources, it's possible to set captions for dimensions and measures. For data from Elasticsearch, it's possible to customize hierarchy captions, formats, time zones, control field visibility, and more.

We recommend using mapping instead of defining a meta-object for JSON (https://www.flexmonster.com/doc/data-types-in-json/) or adding prefixes for CSV (https://www.flexmonster.com/doc/data-types-in-csv/) data since the former presents a powerful way to neatly separate a data source from its representation. Moreover, mapping provides more options than using prefixes for CSV data.

This guide contains the following sections:

- Mapping properties (#mapping-properties)
- Ways to define the mapping (#how-to-define-mapping)
- Other ways to customize field presentation (#alternatives-to-mapping)

- Examples (#examples)

Mapping properties

For each field in the data source, you can set the following properties:

- caption (optional) – String. The hierarchy’s caption.
- type (optional) – String. The field’s data type. Only for "json", "csv", and "api" data source types. type can be:
    - "string" – The field stores string data.
    - "number" – The field stores numerical data. It can be aggregated with all available aggregations.
    - "month" – The field stores months. Note that if the field stores month names only (in either short or full form), the field will be recognized by Flexmonster as a field of the "month" type automatically. If the field contains custom month names, specify its type as "month" explicitly.
    - "weekday" – The field stores days of the week.
    - "date" – The field stores a date. Fields of this type are split into 3 different fields: Year, Month, Day. Only for "json" and "csv" data source types.
    - "year/month/day" – The field stores a date. It’s displayed as a multilevel hierarchy with the following levels: Year > Month > Day. Only for "json" and "csv" data source types.
    - "year/quarter/month/day" – The field is a date. It’s displayed as a multilevel hierarchy with the following levels: Year > Quarter > Month > Day. Only for "json" and "csv" data source types.
    - "date string" – The field stores a date. Fields of this type are represented as strings and can be used in rows, columns, or report filters. The component sorts members of such a field as dates. Fields of the "date string" type can be formatted using the datePattern (https://www.flexmonster.com/api/options-object/#datePattern) option (the default pattern is "dd/MM/yyyy").
    - "datetime" – The field stores a date. You can select fields of this type for values and apply min, max, count, and distinctcount aggregations to them.
      Fields of the "datetime" type can be formatted using the dateTimePattern (https://www.flexmonster.com/api/options-object/#dateTimePattern) option (the default pattern is "dd/MM/yyyy HH:mm:ss").
    - "time" – The field stores time. Fields of this type can be formatted using the timePattern (https://www.flexmonster.com/api/options-object/#timePattern) option (the default pattern is "HH:mm:ss").
    - "id" – The field is an id. This field is not shown in the Field List. Fields of this type can be used for editing data.
    - "property" – The field for setting member properties. This field is not shown in the Field List. For example, it can be used to associate a productID with a product. See an example here (https://jsfiddle.net/flexmonster/nm09d7zh/).
- hierarchy (optional) – String. The hierarchy’s name. When configuring hierarchies, specify this property to mark the field as a level of a hierarchy or as a member property of a hierarchy (in this case, the type parameter should be set to "property"). Only for "json", "csv", and "api" data source types.
- parent (optional) – String. The unique name of the parent level. This property is necessary if the field is a level of a hierarchy and has a parent level. Only for "json", "csv", and "api" data source types.
- folder (optional) – String. The field’s folder. Folders are used to group several fields in the Field List. folder supports nesting via / (e.g., "Folder/Subfolder/"). Only for "json", "csv", and "api" data source types.
- aggregations (optional) — Array of strings. This property represents the list of aggregation functions that can be applied to the current measure.
- filters (optional) – Boolean. This property allows enabling and disabling the UI filters for a specific hierarchy. When set to false, the UI filters are disabled. *Default value: true*.
- visible (optional) – Boolean. When set to false, hides the field from the Field List. Only for "elasticsearch", "csv", and "api" data source types.
- interval (optional) – String. Allows aggregating dates by the given interval. The interval property can be used in the following ways:
    - For the date histogram (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-

aggregations-bucket-datehistogram-aggregation.html) in Elasticsearch. Check out supported time units (https://www.elastic.co/guide/en/elasticsearch/reference/current/common-options.html#date-math). Only for the "elasticsearch" data source type.
    - For the "date string" and "datetime" field types. Supported date intervals are the following: "d" for days, "h" for hours, "m" for minutes, and "s" for seconds (e.g., "1d", "7h", "20m", "30s"). Note that grouping by days starts from 1 January 1970. Only for "csv" and "json" data source types.
- isMeasure (optional) – Boolean. When set to true, the field can be selected only as a measure. The isMeasure property should be used only with the strictDataTypes option (https://www.flexmonster.com/api/options-object/#strictDataTypes) (see an example on JSFiddle (https://jsfiddle.net/flexmonster/b9xy0j5u/)). Only for the "json" data source type. *Default value: false*.
- time_zone (optional) – String. Used for the date histogram (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html). You can specify time zones as either an ISO 8601 UTC offset (e.g., +01:00 or -08:00) or as a time zone ID as specified in the IANA time zone database, such as America/Los_Angeles. Only for the "elasticsearch" data source type. Check out an example here (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html#_time_zone_2).
- format (optional) – String. Used to format different types of date fields. format can be used in the following ways:
    - For the date histogram (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html) in Elasticsearch. The date format/pattern is described in the Elasticsearch documentation (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-daterange-aggregation.html#date-format-pattern). Try a live sample on JSFiddle (https://jsfiddle.net/flexmonster/uvwnrzL0/).
      If the datePattern (/api/options-object/#datePattern) option is defined, format will override its value for the field.
      Only for the "elasticsearch" data source type.
    - For a field of the "date string" type, it allows overriding the datePattern (https://www.flexmonster.com/api/options-object/#datePattern) set in the Options Object. The pattern string for the format is the same as in the datePattern option (i.e., "dd/MM/yyyy"). Only for "json", "csv", and "api" data source types. Learn more about date and time formatting (https://www.flexmonster.com/doc/date-and-time-formatting/).
    - For a field of the "datetime" type, it allows overriding the dateTimePattern (https://www.flexmonster.com/api/options-object/#dateTimePattern) set in the Options Object. The pattern string for the format is the same as in the dateTimePattern option (i.e., "dd/MM/yyyy HH:mm:ss"). Only for "json", "csv", and "api" data source types. Learn more about date and time formatting (https://www.flexmonster.com/doc/date-and-time-formatting/).
    - For a field of the "time" type, it allows overriding the timePattern (https://www.flexmonster.com/api/options-object/#timePattern) set in the Options Object. The pattern string for the format is the same as in the timePattern option (i.e., "HH:mm:ss"). Only for "json", "csv", and "api" data source types. Learn more about date and time formatting (https://www.flexmonster.com/doc/date-and-time-formatting/).
- min_doc_count (optional) – Number. Only for the "elasticsearch" data source type. Used for the date histogram (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html). Can be used to show intervals with empty values (min_doc_count: 0). *Default value: 1 (empty intervals are hidden)*.

Ways to define the mapping

In Flexmonster, there are two ways to define the mapping:

1. As an inline Mapping Object (#inline-mapping).
2. As a URL to a JSON file with the mapping (#mapping-via-url).

As an inline Mapping Object

The inline mapping can be defined as follows:

```
{
    dataSource: {
        filename: "data.csv",
        mapping: {
            "Month": {
                "type": "month"
            },
            "Company Name": {
                "type": "string"
            },
            "Region": {
                "type": "string",
                "hierarchy": "Geography"
            },
            "State": {
                "type": "string",
                "parent": "region",
                "hierarchy": "Geography"
            },
            "Price": {
                "type": "number",
                "caption": "Unit Price"
            }
        }
    }
}
```

See the full code on JSFiddle (https://jsfiddle.net/flexmonster/ks6mvb3q/).

As a URL to a JSON file with the mapping

To define the mapping as a URL to a file, complete the steps below.

Step 1. Create a new JSON file with the mapping (e.g., mapping.json). Its contents should look similar to the following:

```
{
    "Month": {
        "type": "month"
    },
    "Company Name": {
        "type": "string"
    },
    "Region": {
        "type": "string",
        "hierarchy": "Geography"
    },
    "State": {
        "type": "string",
        "parent": "region",
        "hierarchy": "Geography"
    },
    "Price": {
        "type": "number",
        "caption": "Unit Price"
    }
}
```

Step 2. Put your mapping file where Flexmonster can access it.

Step 3. In Flexmonster, define the mapping as a URL to your file:

```
{
    dataSource: {
        filename: "data.csv",
        mapping: "<URL_to_your_mapping_file>"
    }
}
```

See a live sample on JSFiddle (https://jsfiddle.net/flexmonster/5oxfubmc/).

Other ways to customize field presentation

Another way to define how the fields are displayed in the report is by configuring this right in the data source. Please note that this approach is available only for CSV and JSON data sources. For more details, please refer to the Data types in CSV (https://www.flexmonster.com/doc/data-types-in-csv/) and Data types in JSON (https://www.flexmonster.com/doc/data-types-in-json/) articles. It also should be noted that there are certain limitations for the CSV data source – not all the field properties can be customized using prefixes.

For this reason, we strongly recommend using the **Mapping Object** to customize fields.

Examples

1) Creating multilevel hierarchies in JSON:

```
mapping: {
    "Color": {
        type: "string"
    },
    "Country": {
        type: "string",
        hierarchy: "Geography"
    },
    "State": {
        type: "string",
        hierarchy: "Geography",
        parent: "Country"
    },
    "City": {
        type: "string",
        hierarchy: "Geography",
        parent: "State"
    },
    "Price": {
        type: "number"
    }
}
```

Try the live demo on JSFiddle (https://jsfiddle.net/flexmonster/yb20apuL/).

2) Setting custom captions and field data types as well as creating multilevel hierarchies in a CSV data source:

```
mapping: {
    "Order ID": { "type": "string" },
    "Month": { "type": "month" },
    "Company Name": { "type": "string" },
    "Customer": { "type": "string" },
    "region": {
         "caption": "Region",
         "type": "string",
         "hierarchy": "Geography"
     },
     "State": {
         "type": "string",
         "parent": "region",
         "hierarchy": "Geography"
     },
     "City": {
         "type": "string",
         "parent": "State",
         "hierarchy": "Geography"
     },
     "Salesperson": { "type": "string" },
     "Payment Method": { "type": "string" },
     "Category": { "type": "string" },
     "Name": { "type": "string", "caption": "Product Name" },
     "price": { "type": "number", "caption": "Unit Price" },
     "Quantity": { "type": "number" },
     "Revenue": { "type": "number" },
     "Shipping Cost": { "type": "number" }
}
```

Try it on JSFiddle (https://jsfiddle.net/flexmonster/ks6mvb3q/).

3) Setting custom captions and grouping fields in separate folders in a JSON data source:

```
mapping: {
    "Color": {
        caption: "color"
    },
    "Country": {
        caption: "MyCountry",
        folder: "Place"
    },
    "State": {
        caption: "MyState",
        folder: "Place"
    },
    "City": {
        caption: "MyCity",
        folder: "Place"
    },
    "Price": {
        type: "number",
        caption: "MyPrice"
    },
    "Quantity": {
        type: "number",
        caption: "MyQuantity"
    }
}
```

See the live demo on JSFiddle (https://jsfiddle.net/flexmonster/gbjs7tw1/).

4) Specifying custom captions for hierarchies and measures for SSAS:

```
mapping: {
    "[Geography].[Geography]": {
        caption: "My Geography"
    },
    "[Product].[Category]": {
        caption: "My Category"
    }
}
```

See the live demo on JSFiddle (https://jsfiddle.net/flexmonster/s7vndLu8/).

6.6. Number formatting

The way numeric values are formatted in the component can be defined in a report.

The default format is applied to all measures. In addition to this default format, specific number formats can be defined for certain measures. More details can be found in the following sections:

- Number format properties (#properties)
- Default number format (#default)
- Number format for a specific measure (#currency)
- Change number formatting using the Toolbar (#toolbar)
- Number formatting via API (#api)

If the component is connected to an OLAP cube wherein you have already formatted numbers, you can display these formatted values without applying any additional number formatting in the component. More information about this option below:

- Number formatting from an OLAP cube (/doc/number-formatting/#fromcube)

To see live examples on how to format numbers, refer to our Examples page (https://www.flexmonster.com/examples/#!number-formatting).

Number format properties

With number formatting you can define the following:

- Separators for thousands and for decimals.
- The number of decimals to show after the decimal separator.
- The display of null and infinity values.
- The format for currencies specifying both the currency symbol and its position – right or left of the number.

Below is a list of all available properties:

- name – String. It should be unique as it identifies the format in the report. Note: the format with the name property set to "" defines a default number format and it is applied to all the measures without a specific number format. *Default value: ""*.
- thousandsSeparator – String. *Default value: " " (space)*.
- decimalSeparator – String. *Default value: "."*.
- decimalPlaces – Number. The exact number of decimals to show after the decimal separator. When set to -1, the entire number is shown. Note that for e-notation numbers (e.g., 5.8e+23), at least one decimal is always shown after the decimal separator, even if decimalPlaces is set to 0. *Default value: -1*.
- maxDecimalPlaces – Number. The maximum number of decimals to show after the decimal separator. When set to -1, the entire number is shown. Note that for e-notation numbers (e.g., 5.8e+23), at least one decimal is always shown after the decimal separator, even if maxDecimalPlaces is set to 0. Default value: *-1*.
- maxSymbols – Number. The maximum number of symbols in a cell. *Default value: 20*.
- negativeNumberFormat – String. The format of the negative numbers. It can be one of the following values: "-1", "- 1", "1-", "1 -", and "(1)". *Default value: "-1"*.
- currencySymbol – String. The symbol which is shown to the left or the right of the value (e.g. currency symbol, hours, percent, etc.). Learn more about the ways to set the property (#setting-currency-symbol). *Default value: ""*.
- positiveCurrencyFormat – String. The format of the currency symbol. It can be either "$1" or "1$". *Default value: "$1"*.
- negativeCurrencyFormat – String. The format of the currency symbol to display negative amounts. It can be one of the following values: "-$1", "-1$", "$-1", "$1-", "1-$", "1$-", "($1)", and "(1$)". *Default value: "-$1"*.
- isPercent – Boolean. When set to true, data is formatted as a percentage. The behavior is the same as in Excel. Setting isPercent to true will result in numbers being multiplied by 100 and shown with a % symbol. For example, 0.56 gets changed to 56%. Note: if % is set as currencySymbol, setting isPercent to true will not multiply numbers by 100. *Default value: false*.
- nullValue – String. Defines how to show null values in the grid. *Default value: ""*.
- infinityValue – String. Defines how to show infinity values in the grid. *Default value: "Infinity"*.
- divideByZeroValue – String. Defines how to show divided by zero values in the grid. *Default value: "Infinity"*.
- textAlign – String. The alignment of formatted values in cells on the grid. It can have the following values: "right", "left", and "center". *Default value: "right"*.
- beautifyFloatingPoint – Boolean. In JavaScript the output of console.log(.1 + .2);is 0.30000000000000004. Check https://0.30000000000000004.com (https://0.30000000000000004.com) for more details about the problem. When the beautifyFloatingPoint property is set to true, numbers such as 0.30000000000000004 are formatted as 0.3. Setting beautifyFloatingPoint to false means that the full number will be shown. *Default value: true*.

Setting a currency symbol

The currencySymbol property can be set in two ways: by adding the symbol itself or adding a numeric or named HTML code of the symbol. For example, setting "&", "&" or "&" will result in displaying ampersand as a currency symbol.

Both approaches can be used interchangeably unless you are planning to export the report.

If you are going to export the report to PDF, Excel, or CSV, it’s better to add symbols by copy and paste. Otherwise, the symbols will be displayed as code. If you are going to export the report to HTML, you can use both variants as equal.

Default number format

The component has a built-in default number format that is applied to all measures by default. It is composed of the default values of the number format properties (/doc/number-formatting/#properties). The default format can be overridden in a report.

To override the default number format for a report, define a number format with an empty string name property in a report, as follows:

```
{
    dataSource: {
        filename: "data.csv"
    },
    formats: [
        {
            name: "",
            thousandsSeparator: " ",
            decimalSeparator: ".",
            decimalPlaces: -1,
            maxDecimalPlaces: -1,
            maxSymbols: 20,
            negativeNumberFormat: "-1",
            currencySymbol: "",
            negativeCurrencyFormat: "-$1",
            positiveCurrencyFormat: "$1",
            isPercent: false,
            nullValue: "",
            infinityValue: "Infinity",
            divideByZeroValue: "Infinity",
            textAlign: "right",
            beautifyFloatingPoint: true
        }
    ],
    slice: {
        rows: [
            { uniqueName: "Country" }
        ],
        columns: [
            { uniqueName: "[Measures]" }
        ],
        measures: [
            {
                uniqueName: "Price",
                aggregation: "sum",
                active: true
            },
            {
                uniqueName: "Quantity",
                aggregation: "sum",
                active: true
            }
        ]
    }
}
```

See the example on JSFiddle (https://jsfiddle.net/flexmonster/xjav1g36/).

Number format for a specific measure

A number format can be applied to a specific measure or measures. Each measure can have only one format, but a format can be applied to multiple measures.

For example, if you are visualizing financial data, you may want to apply currency formatting to some of the measures in addition to the default format. To apply a format to a specific measure:

1. Name a format.
2. Define the format name for the measure(s) in a default slice.

Properties defined in the default format get applied to all other formats. In the following example each measure with number formats "currency" and "amount" will have thousandsSeparator: ",", since it was defined in the default format:

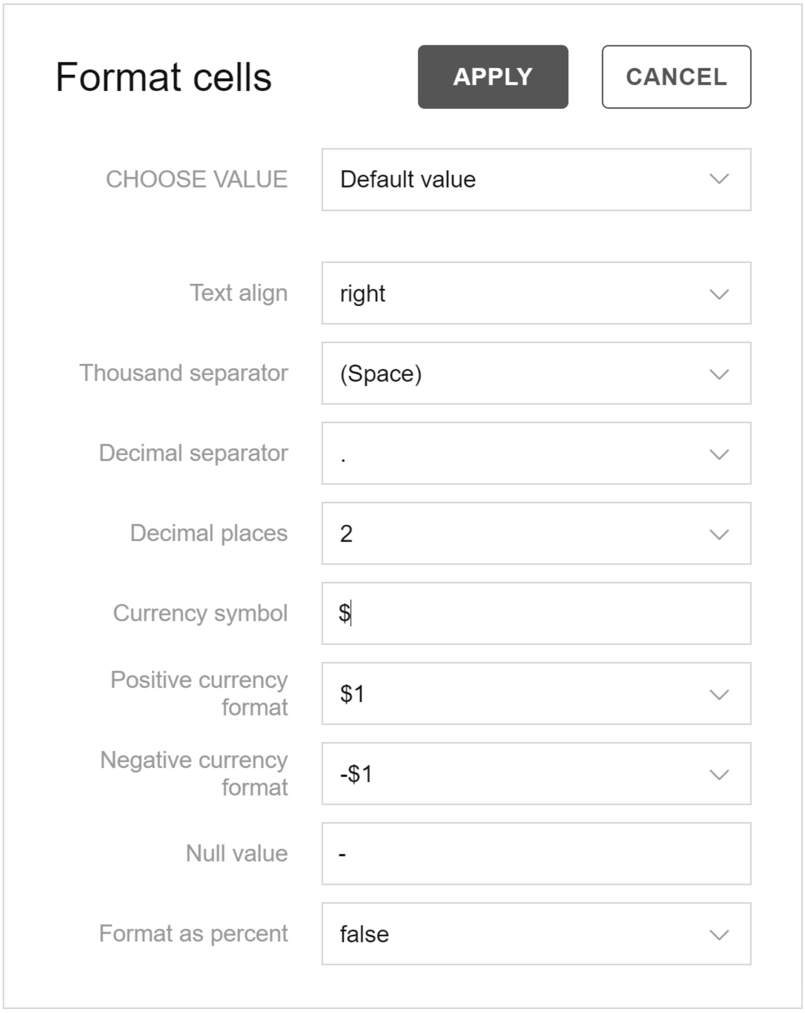
```
{
    dataSource: {
        filename: "https://www.flexmonster.com/download/data.csv"
    },
    formats: [
        {
            name: "",
            thousandsSeparator: ","
        },
        {
            name: "currency",
            currencySymbol: "$"
        },
        {
            name: "amount",
            decimalPlaces: 0,
            currencySymbol: " pcs.",
            positiveCurrencyFormat: "1$"
        }
    ],
    slice: {
        rows: [ { uniqueName: "Category" } ],
        measures: [
            {
                uniqueName: "Price",
                aggregation: "sum",
                active: true,  format: "currency"
            },
            {
                uniqueName: "Discount",
                aggregation: "sum",
                active: false,
                format: "currency"
            },
            {
                uniqueName: "Quantity",
                aggregation: "sum",
                active: true,
                format: "amount"
            }
        ]
    }
}
```

Open the example on JSFiddle (https://jsfiddle.net/flexmonster/qxyse7n6/).

A format can be defined for measure(s) even if they are not active (active property is false) in a default slice.

Change number formatting using the Toolbar

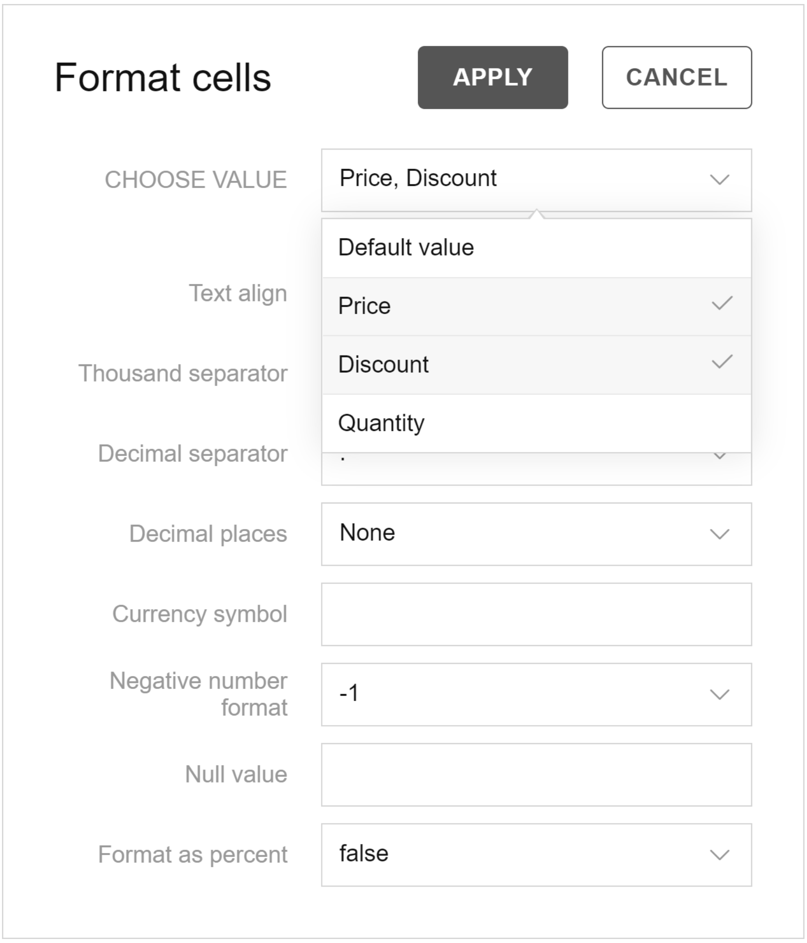
Use Format > Format cells in the Toolbar to change/define number formatting for measures at runtime.



CHOOSE VALUE	Default value
Text align	right
Thousand separator	(Space)
Decimal separator	.
Decimal places	2
Currency symbol	$
Positive currency format	$1
Negative currency format	-$1
Null value	-
Format as percent	false

The number format will be applied to the measures and will be saved within the report.

Starting from version 2.8.22, you can apply formatting to several measures simultaneously. Just choose the needed values in the corresponding dropdown menu:



Number formatting via API

The API calls setFormat() (/api/setformat/) and getFormat() (/api/getformat/) can be used to manipulate number formatting at runtime.

Number formatting from an OLAP cube

If you have already defined formats for measures in an OLAP cube and you want to use those formatted values, set the useOlapFormatting report property to true (it is turned off by default), as follows:

```
{
    dataSource: {
        type: "microsoft analysis services",
        proxyUrl: "https://olap.flexmonster.com/olap/msmdpump.dll",
        cube: "Adventure Works",
        catalog: "Adventure Works DW Standard Edition"
    },
    slice: {
        rows: [
            {uniqueName: "[Product].[Category]"},
            {uniqueName: "[Reseller].[Business Type]"}
        ],
        columns: [{uniqueName: "[Measures]"}],
        measures: [ {uniqueName: "[Measures].[Reseller Order Count]"}]
    },
    options: {
        useOlapFormatting: true
    }
}
```

Check out on JSFiddle (https://jsfiddle.net/flexmonster/npu21mke/).

useOlapFormatting is supported for Microsoft Analysis Services via both Flexmonster Accelerator and XMLA, and for Mondrian via Flexmonster Accelerator. It is not available for Mondrian via XMLA.

6.7. Conditional formatting

Conditional formatting is used to format a cell or a range of cells based on specified criteria. In one report, you can create as many conditions as you need and each condition can apply different formatting rules. Multiple conditional formatting rules for the report will be applied one by one in the order that they were created.

Conditional formatting rules may be added to all pivot table cells, to the cell specifying row and column indexes, to totals and subtotals only, to regular cells only, or to the cells of the selected measure, hierarchy, and hierarchy's member.

Conditions can be defined within a report. When you save the report all the conditional formatting will also be saved and loaded when the report is retrieved.

To see different live examples on how to use conditional formatting, visit our Examples page (https://www.flexmonster.com/examples/#!conditional-formatting).

More details about conditional formatting are available in the following sections:

- Conditional format properties (#properties)
- Style object format (#style)
- Conditional formatting for all values (#values)
- Conditional formatting for specific values (#specific)
- Change conditional formatting using the Toolbar (#toolbar)
- Conditional formatting via API (#api)

Conditional format properties

With conditional formatting you can define the following: a logical expression for the rules of the condition (the formula property); style objects for cells that pass the condition (the format property); and the cells to which the condition is applied. Style objects are composed of font size, font color, font family, and background color.

Here is a list of all available properties for conditions:

- formula – String. A condition that can contain the following logical operators: AND, OR, ==, !=, >, <, >=, <=, +, -, *, /, isNaN(), !isNaN().
  #value is used as a reference to the cell value in the condition. Example: "#value > 2".
  To refer to another field's value in the condition, use the field's name. Example: "'Price' > 2".
- format – Object. The style object that will be applied to a cell if the condition for the cell value is met. Note: when exporting to Excel and PDF, colors should be set to hex color codes.
- formatCSS (optional, read-only) – String. Represents a ready to use CSS string of the format style object. The format style object has properties with names that differ from CSS. The component transforms format to formatCSS.
- id (optional) – String. The id of the conditional formatting rule. If the id property is not set, the id for the rule

will be set inside the pivot component.
- row (optional) – Number. The row index to which the condition should be applied.
- column (optional) – Number. The column index to which the condition should be applied.
- measure (optional) – String. The unique measure name to which the condition should be applied.
- hierarchy (optional) – String. The unique hierarchy name to which the condition should be applied. Must be used with the member property.
- member (optional) – String. The unique member name to which the condition should be applied. Must be used with the hierarchy property.
- isTotal (optional) – Boolean. If it is not defined, the condition will be applied to all cells. If it is set to true, the condition will be applied to total and subtotal cells only. If it is set to false, the condition will be applied to regular cells only.

Style object format

format is a style object that can have the following properties:

```
"format": {
    "backgroundColor": "#FFFFFF",
    "color": "#0000FF",
    "fontFamily": "Arial",
    "fontSize": "12px"
}
```

You can specify only the necessary properties.

If you want to export the pivot table to Excel and PDF, colors should be set to hex color codes.

Conditional formatting for all values

You need to specify the formula and format properties to apply the conditional rule to all values. You can define a format the following way:

```
{
    "dataSource": {
        "filename": "https://www.flexmonster.com/download/data.csv"
    },
    "conditions": [
        {
            "formula": "#value < 400000",
            "format": {
                "backgroundColor": "#FFFFFF",
                "color": "#0000FF",
                "fontFamily": "Arial",
                "fontSize": "12px"
            }
        }
    ]
}
```

Open the example on JSFiddle (https://jsfiddle.net/flexmonster/zypcc8tx/).

Conditional formatting for specific values

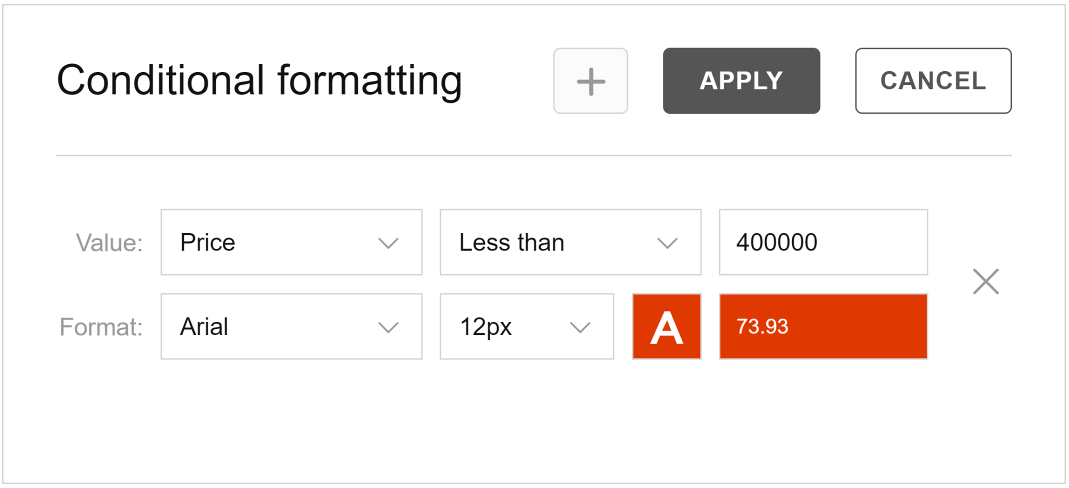
A formatting rule can be applied to a specific measure, hierarchy, hierarchy member, column, or row. Additionally, you can apply it only to regular cells or to totals and subtotals. For example, if you are visualizing financial data, you may want to apply conditional formatting only to regular cells with prices. See an example below:

```
{
    "dataSource": {
        "filename": "https://www.flexmonster.com/download/data.csv"
    },
    "conditions": [
        {
            "formula": "#value < 400000",
            "measure": "Price",
            "isTotal": false,
            "format": {
                "backgroundColor": "#FFFFFF",
                "color": "#0000FF",
                "fontFamily": "Arial",
                "fontSize": "12px"
            }
        }
    ],
    "slice": {
        "rows": [ {"uniqueName": "Category"} ],
        "measures": [
            {"uniqueName": "Price"},
            {"uniqueName": "Quantity"}
        ]
    }
}
```

Open the example on JSFiddle (https://jsfiddle.net/flexmonster/gd7woe0g/).

Change conditional formatting using the Toolbar

Go to Format > Conditional formatting in the Toolbar to change/define conditional formatting rules for values at runtime.



Value: Price | Less than | 400000

Format: Arial | 12px | A | 73.93

This conditional formatting will be applied to the specified values and will be saved within the report.

Conditional formatting via API

The API call addCondition() (/api/addcondition/) is used to add or change a conditional formatting rule at runtime. You can change conditions along with other report parts using the API call setReport() (/api/setreport/).

6.8. Set the report for the component

There are several ways to set the report for the component using JSON:

- Use a report object in the new Flexmonster() API call:

```
var jsonData = [
    {
        "Color" : "green",
        "Country" : "Canada",
        "State" : "Ontario",
        "City" : "Toronto",
        "Price" : 174,
        "Quantity" : 22
    },
    {
        "Color" : "red",
        "Country" : "USA",
        "State" : "California",
        "City" : "Los Angeles",
        "Price" : 166,
        "Quantity" : 19
    }
];

var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    report: {
        dataSource: {
            data: jsonData
        },
        slice: {
            rows: [
                { uniqueName: "Color" },
                { uniqueName: "[Measures]" }
            ],
            columns: [
                { uniqueName: "Country" }
            ],
            measures: [
                { uniqueName: "Price",  aggregation: "sum"  }
            ]
        }
    }
});
```

Check out an example on JSFiddle (https://jsfiddle.net/flexmonster/5bv9bfbr/).

- Use the setReport() API call as shown below:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});

function setReport() {
    var report = flexmonster.getReport();
    // parse, change or save for later use
    flexmonster.setReport(report);
}
```

Check out an example on JSFiddle (https://jsfiddle.net/flexmonster/u981gsL2/).

- Use the open() API call to select a JSON file from the local file system:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});

function openReport() {
    flexmonster.open();
}
```

Check out an example on JSFiddle (https://jsfiddle.net/flexmonster/pzt3ynt1/).

- Use the load() API call to load a JSON file from a URL:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});

function loadReport() {
    flexmonster.load("report.json");
}
```

Check out an example on JSFiddle (https://jsfiddle.net/flexmonster/t78pfd0o/).

Note: XML reports are deprecated starting from version 2.6 but are supported for backward compatibility.

6.9. Get the report from the component

There are several ways to get the report in JSON format from the component:

- You can get the report from the component using the getReport() API call:

  ```
  var pivot = new Flexmonster({
      container: "pivotContainer",
      toolbar: true
  });

  function getReport() {
      var report = flexmonster.getReport();
      // parse, change or save for later use
      flexmonster.setReport(report);
  }
  ```

  Check out an example on JSFiddle (https://jsfiddle.net/flexmonster/8hrmnc9L/).

- Use the save() API call to save a JSON report to the server or to the local file system:

  ```
  var pivot = new Flexmonster({
      container: "pivotContainer",
      toolbar: true
  });

  function saveReport() {
      flexmonster.save("report.json", "file");
  }
  ```

  Check out an example on JSFiddle (https://jsfiddle.net/flexmonster/a21uj95v/).

The easiest way to create a report JSON file is: open the pivot table demo (/demos) that is already connected to the sample CSV data and click the Save button on the Toolbar.

Here is a sample of a JSON report:

```
{
    "dataSource": {
        "type": "microsoft analysis services",
        "proxyUrl": "http://olap.flexmonster.com/olap/msmdpump.dll",
        "dataSourceInfo": "WIN-IA9HPVD1RU5",
        "catalog": "Adventure Works DW Standard Edition",
        "cube": "Adventure Works",
        "binary": false
    },
```

```
    "slice": {
        "rows": [
            {
                "uniqueName": "[Geography].[Geography]",
                "levelName": "[Geography].[Geography].[State-Province]",
                "filter": {
                    "members": [
                        "[Geography].[Geography].[City].&[Malabar]&[NSW]",
                        "[Geography].[Geography].[City].&[Lavender Bay]&[NSW]",
                        "[Geography].[Geography].[City].&[Matraville]&[NSW]",
                        "[Geography].[Geography].[City].&[Milsons Point]&[NSW]"
                    ],
                    "negation": true
                },
                "sort": "asc"
            },
            {
                "uniqueName": "[Sales Channel].[Sales Channel]",
                "sort": "asc"
            }
        ],
        "columns": [
            {
                "uniqueName": "[Measures]"
            }
        ],
        "measures": [
            {
                "uniqueName": "[Measures].[Reseller Order Count]",
                "aggregation": "none",
                "active": true,
                "format": "29mvnel3"
            }
        ],
        "expands": {
            "expandAll": false,
            "rows": [
                {
                    "tuple": [
                        "[Geography].[Geography].[City].&[Lane Cove]&[NSW]"
                    ]
                }
            ]
        },
        "drills": {
            "drillAll": false,
            "rows": [
                {
                    "tuple": [
                        "[Geography].[Geography].[Country].&[Australia]"
                    ]
                },
                {
                    "tuple": [
                        "[Geography].[Geography].[State-Province].&[NSW]&[AU]"
                    ]
                },
                {
                    "tuple": [
                        "[Geography].[Geography].[City].&[Darlinghurst]&[NSW]"
                    ]
                }
            ]
        }
    },
    "options": {
        "viewType": "grid",
        "grid": {
            "type": "compact",
            "title": "",
            "showFilter": true,
            "showHeaders": true,
            "showTotals": "on",
            "showGrandTotals": "on",
            "showExtraTotalLabels": false,
            "showHierarchies": true,
            "showHierarchyCaptions": true,
            "showReportFiltersArea": true
        },
        "chart": {
            "type": "bar",
            "title": "",
            "showFilter": true,
            "multipleMeasures": false,
            "oneLevel": false,
            "autoRange": false,
            "reversedAxes": false,
            "showLegendButton": false,
            "showAllLabels": false,
            "showMeasures": true,
            "showOneMeasureSelection": true,
            "showWarning": true,
            "activeMeasure": ""
        },
        "configuratorActive": false,
        "configuratorButton": true,
        "showAggregations": true,
        "showCalculatedValuesButton": true,
        "editing": false,
        "drillThrough": true,
        "showDrillThroughConfigurator": true,
        "sorting": "on",
        "datePattern": "dd/MM/yyyy",
        "dateTimePattern": "dd/MM/yyyy HH:mm:ss",
        "saveAllFormats": false,
        "showDefaultSlice": true,
        "showEmptyData": false,
        "defaultHierarchySortName": "asc",
        "showOutdatedDataAlert": false
    },
    "conditions": [
        {
            "formula": "#value < 40",
            "format": {
                "backgroundColor": "#FFCCFF",
                "color": "#000033",
                "fontFamily": "Arial",
                "fontSize": "12px"
            }
        }
    ],
    "formats": [
        {
            "name": "29mvnel3",
            "thousandsSeparator": " ",
            "decimalSeparator": ".",
            "decimalPlaces": -1,
            "maxDecimalPlaces": -1,
            "maxSymbols": 20,
            "currencySymbol": "$",
            "negativeCurrencyFormat": "-$1",
            "positiveCurrencyFormat": "$1",
            "nullValue": "",
            "infinityValue": "Infinity",
            "divideByZeroValue": "Infinity",
            "textAlign": "right",
            "isPercent": false
        }
    ],
    "tableSizes": {
        "columns": [
            {
                "tuple": [],
                "measure": "[Measures].[Reseller Order Count]",
                "width": 182
            }
        ]
    },
    "localization": "loc-es.json"
}
```

6.10. Date and time formatting

This tutorial explains how to format date and time fields when using "json", "csv", and "api" data source types.

If your data source type is "microsoft analysis services", you can configure date and time formatting inside the cube and apply it to your data (https://www.flexmonster.com/api/options-object/#useOlapFormatting).

If your data source type is "elasticsearch", follow this guide (/doc/configuring-the-mapping/#set-date-format) to configure date and time formatting.

Follow the steps below to format your date and time fields.

Step 1. Learn about input date and time formats

Flexmonster supports different input formats for date and time fields:

- Input date format (#input-date-format)
- Input time format (#input-time-format)

Step 1.1. Input date format

Input date format depends on your data source:

CSV, JSON, and Flexmonster Data Server

For JSON, CSV, and Flexmonster Data Server data sources, the component supports ISO 8601 (https://www.w3.org/TR/NOTE-datetime) as an input date format:

- "2021-05-25" – Date.
- "2021-05-25T21:30:00" – Date and time.
- "2021-05-25T21:30:00+03:00" – Date and time with a time zone.

Other formats aren’t officially supported and may have unexpected results.

MongoDB

When the MongoDB Connector is used, dates should be defined in the format supported by the MongoDB database (https://docs.mongodb.com/manual/reference/method/Date/). For example, you can define dates in the ISO 8601 (https://www.w3.org/TR/NOTE-datetime) format:

```
{
    "Date": "2021-05-25"
}
```

Custom data source API

If using your implementation of the custom data source API, dates in the initial dataset can have any format. Your custom data source API server should process and pass them to Flexmonster as Unix timestamps (https://www.unixtimestamp.com/). For example, "2021-05-25" will be "1621900800" in the Unix timestamp format.

Step 1.2. Input time format

Time should be specified in seconds when using "json", "csv", or "api" data source types. Here is how it can be done in a JSON data source:

```
{
    "Time": 121
}
```

Note that it is important to define the field’s type as "time" explicitly. Otherwise, the component will treat this field as a numeric one.

For details on setting the field’s type, see the next step (#choose-field-type).

Step 2. Choose a type for your date and time fields

By default, date fields in Flexmonster have the following types:

- "date" – When using "json" and "csv" data source types. Fields of this type are split into three different fields: Year, Month, and Day.
- "date string" – When using the "api" data source type. The component treats fields of this type as strings; they can be used in rows, columns, or report filters.

Time fields have the "number" type and are treated as numbers by default.

Based on your requirements, you can override these default types.

The first way is to use the defaultDateType (https://www.flexmonster.com/api/options-object/#defaultDateType) option. It allows overriding the default date type for "json" and "csv" data source types:

```
{
    options: {
        defaultDateType: "date string"
    }
}
```

Now the "date string" type will be applied to all date fields in the dataset.

Try a live sample on JSFiddle (https://jsfiddle.net/flexmonster/x82zt5oe/).

Another way to define the field type is using the Mapping Object (https://www.flexmonster.com/api/mapping-object/). It provides the following types for date and time fields:

- "date" – Fields of this type are split into three different fields: Year, Month, and Day. Only for "csv" and "json" data source types.
- "year/month/day" – Fields of this type are displayed as a multilevel hierarchy with the following levels: Year > Month > Day. Only for "csv" and "json" data source types.
- "year/quarter/month/day" – Fields of this type are displayed as a multilevel hierarchy with the following levels: Year > Quarter > Month > Day. Only for "csv" and "json" data source types.
- "date string" – Fields of this type are represented as strings and can be used in rows, columns, or report filters. Members of the "date string" fields are sorted as dates.
  Note that Flexmonster does not display time for "date string" fields (i.e., "05/25/2021T21:30:00" will be displayed as "05/25/2021"). To display the time part of a date, use the "datetime" type for the field.
  See how to format the “date string” fields (#format-date-string).
- "datetime" – You can select fields of this type for values and apply min, max, count, and distinctcount aggregations to them.
  See how to format the “datetime” fields (#format-datetime).
- "time" – The field stores time. See how to format the “time” fields (#format-time).

Here is an example of setting the type for a date field using the Mapping Object:

```
dataSource: {
    data: [
        {
            "Date1": "2021-03-04T22:52:00",
            "Date2": "2021-03-04T22:52:00",
            "Date3": "2021-03-04T22:52:00"
        }
    ],
    mapping: {
        "Date1": {
            "type": "date"
        },
        "Date2": {
            "type": "date string"
        },
        "Date3": {
            "type": "datetime"
        }
    }
}
```

See a live sample on JSFiddle (https://jsfiddle.net/flexmonster/nj9cfhzL/).

Step 3. Set date format

In Flexmonster, date fields are formatted using pattern strings.

A pattern consists of letters that are replaced with date and time values in the formatted string. For example, in the pattern "yyyy/MM", the "yyyy" substring is replaced with a four-digit year followed by a "/" character, and the "MM" substring is replaced with a two-digit month.

You can define pattern strings for:

- Fields of the “date string” type (#format-date-string)
- Fields of the “datetime” type (#format-datetime)
- Fields of the “time” type (#format-time)
- Specific fields from the dataset (#format-specific)

Step 3.1. Format fields of the “date string” type

To format fields of the "date string" type, use the datePattern (https://www.flexmonster.com/api/options-object/#datePattern) property of the Options Object. Its default pattern string is "dd/MM/yyyy".

Here is how the datePattern can be specified in the report:

```
{
    options: {
        datePattern: "yyyy.MM.dd"
    },
    dataSource: {
        data: [
            ...
        ],
        mapping: {
            "Date": { "type": "date string" }
        }
    }
}
```

Test or modify this example on JSFiddle (https://jsfiddle.net/flexmonster/tc8yosgd/).

Supported patterns for the "date string" fields are the following:

- d – Day of the month. It is represented as a one- or two-digit number. For example, 2 or 18.
- dd – Day of the month. It is represented as a two-digit number. For example, 02 or 18.
- ddd – Day of the week. It is represented as a three-letter abbreviation of the day of the week. For example, Wed.
- dddd – Day of the week. It is represented as the full name of the day of the week. For example, Wednesday.
- M – Month. It is represented as a one- or two-digit number. For example, 3 or 11.
- MM – Month. It is represented as a two-digit number. For example, 03.
- MMM – Month. It is represented as a three-letter abbreviation of the name of the month. For example, Mar.
- MMMM – Month. It is represented as the full name of the month. For example, March.
- yy – Year. It is represented as a two-digit number. For example, 16.
- yyyy – Year. It is represented as a four-digit number. For example, 2016.
- UTC: – indicates that the UTC time zone should be used. Example format: "UTC:dd/MM/yyyy".
- GMT+-N: – indicates the time zone to use, where N can be changed from 1 to 12. Example format: "GMT+6:dd/MM/yyyy".

Have a look at how the datePattern can be specified:

- "yyyy-MM-dd". Example of a formatted date: 2021-05-25.
- "MMMM d, yyyy". Example of a formatted date: May 25, 2021.

Step 3.2. Format fields of the “datetime” type

To format fields of the "datetime" type, use the dateTimePattern (https://www.flexmonster.com/api/options-object/#dateTimePattern) property of the Options Object. Its default pattern string is "dd/MM/yyyy HH:mm:ss".

Here is how the dateTimePattern can be specified in the report:

```
{
    options: {
        dateTimePattern: "yyyy.MM.dd HH:mm TT"
    },
    dataSource: {
        data: [
            ...
        ],
        mapping: {
            "Date": { "type": "datetime" }
        }
    }
}
```

See the full code on JSFiddle (https://jsfiddle.net/flexmonster/atxersgz/).

Supported patterns for the "datetime" fields are the following:

- d – Day of the month. It is represented as a one or two-digit number. For example, 2 or 18.
- dd – Day of the month. It is represented as a two-digit number. For example, 02 or 18.
- ddd – Day of the week. It is represented as a three-letter abbreviation of the day of the week. For example, Wed.
- dddd – Day of the week. It is represented as the full name of the day of the week. For example, Wednesday.
- M – Month. It is represented as a one or two-digit number. For example, 3 or 11.
- MM – Month. It is represented as a two-digit number. For example, 03.
- MMM – Month. It is represented as a three-letter abbreviation of the name of the month. For example, Mar.
- MMMM – Month. It is represented as the full name of the month. For example, March.
- yy – Year. It is represented as a two-digit number. For example, 16.
- yyyy – Year. It is represented as a four-digit number. For example, 2016.
- h – Hour of the day using the 12-hour format [1 – 12]. It is represented as a one or two-digit number. For example, 1 or 12.
- hh – Hour of the day using the 12-hour format [1 – 12]. It is represented as a two-digit number. For example, 01 or 12.
- H – Hour of the day using the 24-hour format [0 – 23]. It is represented as a one or two-digit number. For example, 0 or 23.
- HH – Hour of the day using the 24-hour format [0 – 23]. It is represented as a two-digit number. For example, 00 or 23.
- k – Hour of the day using the 24-hour format [1 – 24]. It is represented as a one or two-digit number. For example, 1 or 24.
- kk – Hour of the day using the 24-hour format [1 – 24]. It is represented as a two-digit number. For example, 01 or 24.
- m – Minutes [0 – 59]. It is represented as a one or two-digit number. For example, 0 or 59.
- mm – Minutes [0 – 59]. It is represented as a two-digit number. For example, 00 or 59.
- s – Seconds [0 – 59]. It is represented as a one or two-digit number. For example, 0 or 59.
- ss – Seconds [0 – 59]. It is represented as a two-digit number. For example, 00 or 59.
- l – Milliseconds. It is represented as a three-digit number. For example, 100.
- L – 10 Milliseconds. It is represented as a two-digit number (rounded, if needed). For example, 10.
- t – am/pm – a one-letter indicator. For example, a or p.
- tt – am/pm – a two-letter indicator. For example, am or pm.
- T – AM/PM – a one-letter indicator. For example, A or P.
- TT – AM/PM – a two-letter indicator. For example, AM or PM.
- UTC: – indicates that the UTC time zone should be used. Example format: "UTC:dd/MM/yyyy HH:mm:ss".
- GMT+-N: – indicates the time zone to use, where N can be changed from 1 to 12. Example format: "GMT+6:dd/MM/yyyy HH:mm:ss".

Have a look at how the dateTimePattern can be specified:

- "yyyy-MM-dd". Example of a formatted date: 2021-05-25.
- "yyyy-MM-dd'T'HH:mm:ss". Example of a formatted date: 2021-05-25T21:30:00.
- "MMMM d, yyyy". Example of a formatted date: May 25, 2021.
- "h:mm TT". Example of a formatted date: 9:30 PM.

- "HH:mm:ss". Example of a formatted date: 21:30:00.
- "h:mm:ss TT". Example of a formatted date: 9:30:00 PM.

Step 3.3. Format fields of the “time” type

To format fields of the "time" type, use the timePattern (https://www.flexmonster.com/api/options-object/#timePattern) property of the Options Object. Its default pattern string is "HH:mm:ss".

Here is how the timePattern can be specified in the report:

```
{
    options: {
        timePattern: "d'd' HH'h' mm'min'"
    },
    dataSource: {
        data: [
            ...
        ],
        mapping: {
            "Time": { "type": "time" }
        }
    }
}
```

Check out a full example on JSFiddle (https://jsfiddle.net/flexmonster/3790gnsm/).

Supported patterns for the "time" fields are the following:

- d – Days. It is represented as a one- or two-digit number. For example, 5 or 11.
- dd – Days. It is represented as a two-digit number. For example, 05 or 11.
- H – Hours. It is represented as a one- or two-digit number. For example, 6 or 15.
- HH – Hours. It is represented as a two-digit number. For example, 06 or 15.
- HHH – Hours. Displays time data in hours even when the number of hours is greater than 24. For example, 46 or 15.
- m – Minutes. It is represented as a one- or two-digit number. For example, 2 or 10.
- mm – Minutes. It is represented as a two-digit number. For example, 02 or 10.
- s – Seconds. It is represented as a one- or two-digit number. For example, 3 or 16.
- ss – Seconds. It is represented as a two-digit number. For example, 03 or 16.

Have a look at how the timePattern can be specified:

- "HH:mm:ss". Example of a formatted time: 21:30:00.
- "h:mm TT". Example of a formatted time: 9:30 PM.
- "h:mm:ss TT". Example of a formatted time: 9:30:00 PM.

Step 3.4. Format specific fields

Formatting specified in the Options Object applies to all the fields of the respective type (e.g., all "date string" fields are formatted according to the datePattern (https://www.flexmonster.com/api/options-object/#datePattern)). To format a specific field, use the format (https://www.flexmonster.com/api/mapping-object/#format) property of the Mapping Object.

It works the same way as datePattern, dateTimePattern, and timePattern options and overrides their values. Here is an example of how to define formatting for a certain field:

```
mapping: {
    "Date": {
        type: "date string",
        format: "MM/dd/yyyy"
    }
}
```

Try a live sample on JSFiddle (https://jsfiddle.net/flexmonster/jeqf23ku/).

As you can see, datePattern is not applied to the Date field since it has the format property in the mapping.

Step 4. Manage time zones

By default, Flexmonster uses the user’s local time zone in dates.

To use another time zone, include "GMT+-N:" at the beginning of the pattern (e.g., "GMT+6:dd/MM/yyyy HH:mm:ss"). N can be changed from 1 to 12. You can set a time zone in both datePattern (https://www.flexmonster.com/api/options-object/#datePattern) and dateTimePattern (https://www.flexmonster.com/api/options-object/#dateTimePattern) options.

Setting a time zone affects only date representation in the component; the original data is not changed. Check out a live sample on JSFiddle (https://jsfiddle.net/flexmonster/dn38gs0q/).

If you need to set a time zone for hierarchical dates (i.e., for "date", "year/month/day", and "year/quarter/month/day" date types), use the options.dateTimezoneOffset (https://www.flexmonster.com/api/options-object/#dateTimezoneOffset) property. See a live demo on JSFiddle (https://jsfiddle.net/flexmonster/r8dk7v63/).

What’s next?

You may be interested in the following articles:

- How to set specific options common for all reports (/doc/global-object/)
- How to configure the way that data is exported (/doc/export-and-print/)
- How to customize the Toolbar (/doc/customizing-toolbar/)
- How to add localization (/doc/localizing-component/)

6.11. Configuring global options

Flexmonster has a wide variety of configurable options: localization, grid options, chart options, etc.

Report objects are used to configure which data should be shown in the component, how it will be visualized, and what instruments will be available for data analysis. The values for all the possible options can be defined in the report. Details on how to configure all options in the report are described in the configuring report set of articles (/doc/configuring-report/).

It is not mandatory to explicitly set all the options in the report. When a value for an option is not set in the report, it is pulled from the default values that are predefined in the component.

It is however often simpler to modify specific default options that will be common for all reports. This can be done using the global object. global is a parameter of the new Flexmonster() (/api/new-flexmonster/) method. In general, the global object should be considered as a global parent report. It can have a dataSource, options, and localization sub-objects of the same structure as the report object (/api/report-object/). The global object overrides the default option values for all the reports of the component. Thus, if a specific option is not explicitly set in the report, the component will use the global value.

Examples

1. Specify one localization for all reports:

   ```
   global: {
     localization: "loc/es.json"
   }
   ```

   Check out an example on JSFiddle (//jsfiddle.net/flexmonster/r36nyjqL/).
2. This example demonstrates how to set common options for all reports:

   ```
   global: {
     options: {
       grid: {
         showFilter: false
       },
       configuratorButton: false,
       sorting: false,
       drillThrough: false
     }
   }
   ```

   Check out an example on JSFiddle (//jsfiddle.net/flexmonster/ge35b4fk/).
3. Apply date and time formatting for “date string” date fields in the component:

   ```
   global: {
     options: {
       datePattern: "'date: 'MM/dd/yy';
       time: 'HH-mm"
     }
   }
   ```

   Check out an example on JSFiddle (//jsfiddle.net/flexmonster/tqmxppj7/).
4. The next example shows how you can specify the dataSource for all reports:

   ```
   global: {
     dataSource: {
       filename: "https://cdn.flexmonster.com/data/data.csv"
     }
   }
   ```

   Check out an example on JSFiddle (//jsfiddle.net/flexmonster/ufv83nvf/).
5. The current report is usually obtained via the save() or getReport() API calls. By default, reports returned by these methods contain only the options that were explicitly defined inside the report object. For example, if we have the component defined like this:

```
var pivot = new Flexmonster({
  container: "pivotContainer",
  global: {
    localization: "loc/es.json"
  },
  report: {
    dataSource: {
      filename: "https://cdn.flexmonster.com/data/data.csv"
    }
  }
});
```

save() and getReport() will get the report containing the specified dataSource, but without any options defined in global or defaults. Use the withGlobals: true parameter in the API calls to include localization that was defined in global in the report:

save()

```
flexmonster.save({
  withGlobals: true
});
```

getReport()

```
flexmonster.getReport({
  withGlobals: true
});
```

To include the default options in the report, use the withDefaults: true parameter:

save()

```
flexmonster.save({
  withDefaults: true
});
```

getReport()

```
flexmonster.getReport({
  withDefaults: true
});
```

Check out an example on JSFiddle (//jsfiddle.net/flexmonster/p9keuua4/).

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (/doc/customizing-toolbar/)
- How to customize appearance (/doc/customizing-appearance/)
- How to add localization (/doc/localizing-component/)
- How to define a format for date and time (/doc/date-and-time-formatting/)
- How to configure export (/doc/export-and-print/)

6.12. Export and print

All current data from the grid or chart can be exported in various formats. It is an easy and convenient way to save the results of your work. The exported file can be saved to the local file system or to your server.

The API call exportTo() (/api/exportto/) is used for exporting and the API call print() (/api/print/) is used for printing. To see different examples of exporting and printing, visit our Examples page (https://www.flexmonster.com/examples/#!export-and-print).

- Print (#print)
- Export to HTML (#html)
- Export to CSV (#csv)
- Export to Excel (#excel)
- Export to an image (#image)
- Export to PDF (#pdf)
- How to export to the server without using a browser (#export-without-browser)

Print

The printing of the contents of the grid or chart via the OS print manager can be configured with the following options object inside the print() call:

options (optional) – Object. Can contain the following print properties:

- header (from v2.211) (optional) – String. The header is set using the HTML format (tags, inline styles, img with base64 src), rendered in the browser, and added as an image to the printed PDF file. The following tokens can be used: ##CURRENT-DATE##, ##PAGE-NUMBER##. They will be replaced by the appropriate data.
- footer (from v2.211) (optional) – String. The footer is set using the HTML format (tags, inline styles, img with base64 src), rendered in the browser, and added as an image to the printed PDF file. The following tokens can be used: ##CURRENT-DATE##, ##PAGE-NUMBER##. They will be replaced by appropriate data.

Below is an example with header and footer parameters:

```
var options = {
    header:"<div>##CURRENT-DATE##</div>",
    footer:"<div>##PAGE-NUMBER##</div>"
}
flexmonster.print(options);
```

Check out how this sample works on JSFiddle (https://jsfiddle.net/flexmonster/pq18xucm/).

Export to HTML

To export to HTML set the first parameter of exportTo() to "html". The second parameter is an optional object that can contain the following properties:

- filename (optional) – String. The name of the created file. *Default name: "pivotgrid".*
- destinationType (optional) – String. Defines where the component's contents will be exported. The destination type can be the following:
  - "file" (default) – the component's contents will be exported to a file to the local computer.
  - "server" – the component's contents will be exported to the server (a server-side script is required). When exporting to a server, Flexmonster creates an XMLHttpRequest (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) and sends it as a POST request.
  - "plain" – the component's contents will be exported as a string and returned with callbackHandler.
- footer (from v2.211) (optional) – String. The footer can be also set using the HTML format (tags, inline styles, img with base64 src). The following token can be used: ##CURRENT-DATE##. It will be replaced by appropriate data.
- header (from v2.211) (optional) – String. The header can be also set using the HTML format (tags, inline styles, img with base64 src). The following token can be used: ##CURRENT-DATE##. It will be replaced by appropriate data.
- url (optional) – String. To save the file to the server, provide the component with a path to the server-side script that can save this file.

The third parameter of exportTo() is an optional callbackHandler. It is a function that is called when the data is ready to be exported. The callbackHandler takes two parameters: result and error objects. Learn more about them in the API reference (https://www.flexmonster.com/api/exportto/#!callbackHandler).

Here is an example of how to save to a local HTML file:

```
var params = {
    filename: 'flexmonster',
    header:"##CURRENT-DATE##",
    footer:"<div style='color:#df3800'>Table Footer</div>",
};
flexmonster.exportTo('html', params);
```

Try the example on JSFiddle (https://jsfiddle.net/flexmonster/45e9k4c1/).

Saving to server:

```
var params = {
    destinationType: 'server',
    url: 'your server-side script'
};
flexmonster.exportTo('html', params);
```

Export to CSV

To export to CSV set the first parameter of exportTo() to "csv". The second parameter is an optional object that can contain the following properties:

- alwaysEnclose (optional) – Boolean. Indicates whether to enclose all CSV fields in quotes. When set to true, the fields are always enclosed in quotes. Otherwise, they will be enclosed only when necessary (e.g., if a field contains a comma: Bike, "$15,000", blue). *Default value: false*.
- fieldSeparator (optional) – String. Defines the field separator to split each CSV row in the export file. *Default separator: ,*.
- filename (optional) – String. The name of the created file. *Default name: "pivot"*.
- destinationType (optional) – String. Defines where the component’s contents will be exported. The destination type can be the following:
    - "file" (default) – the component’s contents will be exported to a file to the local computer.
    - "server" – the component’s contents will be exported to the server (a server-side script is required). When exporting to a server, Flexmonster creates an XMLHttpRequest (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) and sends it as a POST request.
    - "plain" – the component’s contents will be exported as a string and returned with callbackHandler.
- footer (from v2.7.24) (optional) – String. There’s an option to apply the multirow footer in the following way: "Row1\nRow2\nRow3".
- header (from v2.7.24) (optional) – String. There’s an option to apply the multirow header in the following way: "Row1\nRow2\nRow3".
- url (optional) – String. To save the file to the server, provide the component with a path to the server-side script that can save this file.

The third parameter of exportTo() is an optional callbackHandler. It is a function that is called when the data is ready to be exported. The callbackHandler takes two parameters: result and error objects. Learn more about them in the API reference (https://www.flexmonster.com/api/exportto/#!callbackHandler).

Here is how to save a local CSV file and get the resulting data within the callbackHandler:

```
var params = {
    filename: 'flexmonster',
    header:"First header row\nSecond header row",
    footer:"Table Footer",
};
flexmonster.exportTo('csv', params, function(result) {
    console.log(result.data)
});
```

Check out an example on JSFiddle (https://jsfiddle.net/flexmonster/45e9k4c1/).

Saving to server:

```
var params = {
    destinationType: 'server',
    url: 'your server-side script'
};
flexmonster.exportTo('csv', params);
```

Export to Excel

To export to Excel set the first parameter of exportTo() to "excel". The second parameter is an optional object that can contain the following properties:

- filename (optional) – String. The name of the created file. *Default name: "pivot"*.
- destinationType (optional) – String. Defines where the component's contents will be exported. The destination type can be the following:
  - "file" (default) – the component's contents will be exported to a file to the local computer.
  - "server" – the component's contents will be exported to the server (a server-side script is required). When exporting to a server, Flexmonster creates an XMLHttpRequest (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) and sends it as a POST request.
  - "plain" – the component's contents will be exported as a Uint8Array and returned with callbackHandler.
- excelSheetName (from v2.2) (optional) – String. Sets the sheet name.
- footer (from v2.7.24) (optional) – String. There's an option to apply the multirow footer in the following way: "Row1\nRow2\nRow3".
- header (from v2.7.24) (optional) – String. There's an option to apply the multirow header in the following way: "Row1\nRow2\nRow3".
- showFilters (from v2.1) (optional) – Boolean. Indicates whether the filters info will be shown (true) in the exported Excel file or not (false). *Default value: false*.
- url (optional) – String. To save the file to the server, provide the component with a path to the server-side script that can save this file.
- useOlapFormattingInExcel (from v2.2) (optional) – Boolean. Indicates how to export the grid cells to the Excel file if the formatting in the component is taken from an OLAP cube – as a formatted string (true) or as numbers without any formatting (false). In previous versions, this was not configurable and the cells were exported as formatted strings.
- useCustomizeCellForData (optional) – Boolean. Specifies how cells modified by customizeCell are exported: as formatted strings (true) or as numbers without formatting (false). *Default value:* true.

The third parameter of exportTo() is an optional callbackHandler. It is a function that is called when the data is ready to be exported. The callbackHandler takes two parameters: result and error objects. Learn more about them in the API reference (https://www.flexmonster.com/api/exportto/#!callbackHandler).

This is how to save a report as a local Excel file:

```
var params = {
    filename: 'flexmonster',
    header:"First header row\nSecond header row",
    footer:"Table Footer",
    excelSheetName: 'Report',
    showFilters: true,
    useOlapFormattingInExcel: false
};
flexmonster.exportTo('excel', params);
```

Check out an example on JSFiddle (https://jsfiddle.net/flexmonster/45e9k4c1/).

Saving to the server:

```
var params = {
    destinationType: 'server',
    url: 'your server-side script'
};
flexmonster.exportTo('excel', params);
```

Export to an image

To export to an image set the first parameter of exportTo() to "image". The second parameter is an optional object that can contain the following properties:

- filename (optional) – String. The name of the created file. *Default name: "pivotgrid"*.
- destinationType (optional) – String. Defines where the component's contents will be exported. The destination type can be the following:
  - "file" (default) – the component's contents will be exported to a file to the local computer.
  - "server" – the component's contents will be exported to the server (a server-side script is required). When exporting to a server, Flexmonster creates an XMLHttpRequest (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) and sends it as a POST request.
  - "plain" – the component's contents will be exported as an HTMLCanvasElement and returned with callbackHandler.
- footer (from v2.7.24) (optional) – String. The footer can be also set using the HTML format (tags, inline styles, img with base64 src). The following token can be used: ##CURRENT-DATE##. It will be replaced by appropriate data.
- header (from v2.7.24) (optional) – String. The header can be also set using the HTML format (tags, inline styles, img with base64 src). The following token can be used: ##CURRENT-DATE##. It will be replaced by appropriate data.
- url (optional) – String. To save the file to the server, provide the component with a path to the server-side script that can save this file.

The third parameter of exportTo() is an optional callbackHandler. It is a function that is called when the data is ready to be exported. The callbackHandler takes two parameters: result and error objects. Learn more about them in the API reference (https://www.flexmonster.com/api/exportto/#!callbackHandler).

Here is how to save the image as a local file:

```
var params = {
    filename: 'flexmonster',
    header:"##CURRENT-DATE##",
    footer: "<div style='color:#df3800'>Table Footer</div>"
};
flexmonster.exportTo('image', params);
```

Try the example on JSFiddle (https://jsfiddle.net/flexmonster/45e9k4c1/).

Saving to server:

```
var params = {
    destinationType: 'server',
    url: 'your server-side script'
};
flexmonster.exportTo('image', params);
```

Export to PDF

To export to PDF set the first parameter of exportTo() to "pdf". The second parameter is an optional object that can contain the following properties:

- filename (optional) – String. The name of the created file. *Default name: "pivot"*.
- destinationType (optional) – String. Defines where the component's contents will be exported. The destination type can be the following:
    - "file" (default) – the component's contents will be exported to a file to the local computer.
    - "server" – the component's contents will be exported to the server (a server-side script is required). When exporting to a server, Flexmonster creates an XMLHttpRequest (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) and sends it as a POST request.
    - "plain" – this destination type allows you to modify the generated PDF file. The component's contents will be exported to a jsPDF object (https://github.com/MrRio/jsPDF) and returned with the callbackHandler. jsPDF is a library that generates PDFs using client-side JavaScript. After exporting from Flexmonster, the jsPDF object can be modified using the jsPDF API (https://rawgit.com/MrRio/jsPDF/master/docs/) and then saved. See an example (https://jsfiddle.net/flexmonster/qhkvmhn5/).
- fontUrl (from v2.7.7) (optional) – String. The URL to the TTF font file for saving PDF reports in Chinese, Arabic or any other language. Check out the list of ready-to-use Google Noto Fonts (#cdn-fonts-list) that you can use to support almost any language in the world. Only fonts in standard TTF format are supported.
- footer (from v2.211) (optional) – String. The footer can be also set using the HTML format (tags, inline styles, img with base64 src), rendered in the browser, and added as an image to the exported PDF file. The following tokens can be used: ##CURRENT-DATE##, ##PAGE-NUMBER##. They will be replaced by appropriate data.
- header (from v2.211) (optional) – String. The header can be also set using the HTML format (tags, inline styles, img with base64 src), rendered in the browser, and added as an image to the exported PDF file. The following tokens can be used: ##CURRENT-DATE##, ##PAGE-NUMBER##. They will be replaced by appropriate data.
- pageFormat (optional) – String. Defines the page format for the PDF file. The following sizes are available: "A0", "A1", "A2", "A3", "A4", "A5". *Default value: "A4"*.
- pageOrientation (optional) – String. Defines the page orientation for the PDF file. Page orientation can be either "portrait" or "landscape". *Default value: "portrait"*.
- url (optional) – String. To save the file to the server, provide the component with a path to the server-side script that can save this file.

The third parameter of exportTo() is an optional callbackHandler. It is a function that is called when the data is ready to be exported. The callbackHandler takes two parameters: result and error objects. Learn more about them in the API reference (https://www.flexmonster.com/api/exportto/#!callbackHandler).

Here is how to save the report as a local PDF file:

```
var params = {
    filename: 'flexmonster',
    header:"##CURRENT-DATE##",
    footer:"<div>##PAGE-NUMBER##</div>",
    pageOrientation: 'landscape'
};
flexmonster.exportTo('pdf', params);
```

Check out the example on JSFiddle (https://jsfiddle.net/flexmonster/45e9k4c1/).

Saving to server:

```
var params = {
    destinationType: 'server',
    url: 'your server-side script'
};
flexmonster.exportTo('pdf', params);
```

Modifying generated PDF and saving locally:

```
flexmonster.exportTo("pdf", { destinationType: "plain" },
    function(res) {
        var pdf = res.data;
        pdf.addPage();
        pdf.text('Hello world!', 10, 10);
        pdf.save(res.filename);
    }
);
```

Open on JSFiddle (https://jsfiddle.net/flexmonster/qhkvmhn5/).

Setting TTF font file:

```
flexmonster.exportTo('pdf', {
  fontUrl: 'https://cdn.flexmonster.com/fonts/NotoSansCJKtc-Regular.ttf'
});
```

CDN fonts for export

Expand the list of Google Noto Fonts

Language	URL
582 languages	https://cdn.flexmonster.com/fonts/NotoSans-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSans-Regular.ttf)
Adlam	https://cdn.flexmonster.com/fonts/NotoSansAdlam-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSans Adlam-Regular.ttf)
Adlam Unjoined	https://cdn.flexmonster.com/fonts/NotoSansAdlamUnjoin

	ed-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansAdlamUnjoined-Regular.ttf)
Anatolian Hieroglyphs	https://cdn.flexmonster.com/fonts/NotoSansAnatolianHieroglyphs-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansAnatolianHieroglyphs-Regular.ttf)
Arabic	https://cdn.flexmonster.com/fonts/NotoSansArabic-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansArabic-Regular.ttf)
Arabic UI	https://cdn.flexmonster.com/fonts/NotoSansArabicUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansArabicUI-Regular.ttf)
Armenian	https://cdn.flexmonster.com/fonts/NotoSansArmenian-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansArmenian-Regular.ttf)
Avestan	https://cdn.flexmonster.com/fonts/NotoSansAvestan-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansAvestan-Regular.ttf)
Balinese	https://cdn.flexmonster.com/fonts/NotoSansBalinese-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansBalinese-Regular.ttf)
Bamum	https://cdn.flexmonster.com/fonts/NotoSansBamum-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansBamum-Regular.ttf)
Batak	https://cdn.flexmonster.com/fonts/NotoSansBatak-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansBatak-Regular.ttf)
Bengali	https://cdn.flexmonster.com/fonts/NotoSansBengali-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansBengali-Regular.ttf)
Bengali UI	https://cdn.flexmonster.com/fonts/NotoSansBengaliUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansBengaliUI-Regular.ttf)
Brahmi	https://cdn.flexmonster.com/fonts/NotoSansBrahmi-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansBrahmi-Regular.ttf)
Buginese	https://cdn.flexmonster.com/fonts/NotoSansBuginese-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansBuginese-Regular.ttf)
Buhid	https://cdn.flexmonster.com/fonts/NotoSansBuhid-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansBuhid-Regular.ttf)
Japanese	https://cdn.flexmonster.com/fonts/NotoSansCJKjp-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansCJKjp-Regular.ttf)
Korean	https://cdn.flexmonster.com/fonts/NotoSansCJKkr-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansCJKkr-Regular.ttf)
Simplified Chinese	https://cdn.flexmonster.com/fonts/NotoSansCJKsc-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansCJKsc-Regular.ttf)
Traditional Chinese	https://cdn.flexmonster.com/fonts/NotoSansCJKtc-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansCJKtc-Regular.ttf)
Canadian Aboriginal	https://cdn.flexmonster.com/fonts/NotoSansCanadianAboriginal-Regular.ttf (https://cdn.flexmonster.com/fonts/No

	toSansCanadianAboriginal-Regular.ttf)
Carian	https://cdn.flexmonster.com/fonts/NotoSansCarian-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansCarian-Regular.ttf)
Chakma	https://cdn.flexmonster.com/fonts/NotoSansChakma-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansChakma-Regular.ttf)
Cham	https://cdn.flexmonster.com/fonts/NotoSansCham-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansCham-Regular.ttf)
Cherokee	https://cdn.flexmonster.com/fonts/NotoSansCherokee-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansCherokee-Regular.ttf)
Coptic	https://cdn.flexmonster.com/fonts/NotoSansCoptic-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansCoptic-Regular.ttf)
Cuneiform	https://cdn.flexmonster.com/fonts/NotoSansCuneiform-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansCuneiform-Regular.ttf)
Cypriot	https://cdn.flexmonster.com/fonts/NotoSansCypriot-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansCypriot-Regular.ttf)
Deseret	https://cdn.flexmonster.com/fonts/NotoSansDeseret-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansDeseret-Regular.ttf)
Devanagari	https://cdn.flexmonster.com/fonts/NotoSansDevanagari-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansDevanagari-Regular.ttf)
Devanagar iUI	https://cdn.flexmonster.com/fonts/NotoSansDevanagariUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansDevanagariUI-Regular.ttf)
Display	https://cdn.flexmonster.com/fonts/NotoSansDisplay-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansDisplay-Regular.ttf)
Egyptian Hieroglyphs	https://cdn.flexmonster.com/fonts/NotoSansEgyptianHieroglyphs-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansEgyptianHieroglyphs-Regular.ttf)
Ethiopic	https://cdn.flexmonster.com/fonts/NotoSansEthiopic-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansEthiopic-Regular.ttf)
Georgian	https://cdn.flexmonster.com/fonts/NotoSansGeorgian-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansGeorgian-Regular.ttf)
Glagolitic	https://cdn.flexmonster.com/fonts/NotoSansGlagolitic-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansGlagolitic-Regular.ttf)
Gothic	https://cdn.flexmonster.com/fonts/NotoSansGothic-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansGothic-Regular.ttf)
Gujarati	https://cdn.flexmonster.com/fonts/NotoSansGujarati-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansGujarati-Regular.ttf)
Gujarati UI	https://cdn.flexmonster.com/fonts/NotoSansGujaratiUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansGujaratiUI-Regular.ttf)

Gurmukhi	https://cdn.flexmonster.com/fonts/NotoSansGurmukhi-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansGurmukhi-Regular.ttf)
Gurmukhi UI	https://cdn.flexmonster.com/fonts/NotoSansGurmukhiUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansGurmukhiUI-Regular.ttf)
Hanunoo	https://cdn.flexmonster.com/fonts/NotoSansHanunoo-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansHanunoo-Regular.ttf)
Hebrew	https://cdn.flexmonster.com/fonts/NotoSansHebrew-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansHebrew-Regular.ttf)
Imperial Aramaic	https://cdn.flexmonster.com/fonts/NotoSansImperialAramaic-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansImperialAramaic-Regular.ttf)
Inscriptional Pahlavi	https://cdn.flexmonster.com/fonts/NotoSansInscriptionalPahlavi-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansInscriptionalPahlavi-Regular.ttf)
Inscriptional Parthian	https://cdn.flexmonster.com/fonts/NotoSansInscriptionalParthian-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansInscriptionalParthian-Regular.ttf)
Javanese	https://cdn.flexmonster.com/fonts/NotoSansJavanese-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansJavanese-Regular.ttf)
Kaithi	https://cdn.flexmonster.com/fonts/NotoSansKaithi-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansKaithi-Regular.ttf)
Kannada	https://cdn.flexmonster.com/fonts/NotoSansKannada-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansKannada-Regular.ttf)
Kannada UI	https://cdn.flexmonster.com/fonts/NotoSansKannadaUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansKannadaUI-Regular.ttf)
Kayah Li	https://cdn.flexmonster.com/fonts/NotoSansKayahLi-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansKayahLi-Regular.ttf)
Kharoshthi	https://cdn.flexmonster.com/fonts/NotoSansKharoshthi-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansKharoshthi-Regular.ttf)
Khmer	https://cdn.flexmonster.com/fonts/NotoSansKhmer-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansKhmer-Regular.ttf)
Khmer UI	https://cdn.flexmonster.com/fonts/NotoSansKhmerUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansKhmerUI-Regular.ttf)
Lao	https://cdn.flexmonster.com/fonts/NotoSansLao-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansLao-Regular.ttf)
Lao UI	https://cdn.flexmonster.com/fonts/NotoSansLaoUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansLaoUI-Regular.ttf)
Lepcha	https://cdn.flexmonster.com/fonts/NotoSansLepcha-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansLepcha-Regular.ttf)
Limbu	https://cdn.flexmonster.com/fonts/NotoSansLimbu-

	Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansLimbu-Regular.ttf)
Linear B	https://cdn.flexmonster.com/fonts/NotoSansLinearB-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansLinearB-Regular.ttf)
Lisu	https://cdn.flexmonster.com/fonts/NotoSansLisu-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansLisu-Regular.ttf)
Lycian	https://cdn.flexmonster.com/fonts/NotoSansLycian-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansLycian-Regular.ttf)
Lydian	https://cdn.flexmonster.com/fonts/NotoSansLydian-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansLydian-Regular.ttf)
Malayalam	https://cdn.flexmonster.com/fonts/NotoSansMalayalam-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansMalayalam-Regular.ttf)
Malayalam UI	https://cdn.flexmonster.com/fonts/NotoSansMalayalamUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansMalayalamUI-Regular.ttf)
Mandaic	https://cdn.flexmonster.com/fonts/NotoSansMandaic-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansMandaic-Regular.ttf)
Meetei Mayek	https://cdn.flexmonster.com/fonts/NotoSansMeeteiMayek-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansMeeteiMayek-Regular.ttf)
Mongolian	https://cdn.flexmonster.com/fonts/NotoSansMongolian-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansMongolian-Regular.ttf)
Myanmar	https://cdn.flexmonster.com/fonts/NotoSansMyanmar-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansMyanmar-Regular.ttf)
Myanmar UI	https://cdn.flexmonster.com/fonts/NotoSansMyanmarUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansMyanmarUI-Regular.ttf)
N’Ko	https://cdn.flexmonster.com/fonts/NotoSansNKo-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansNKo-Regular.ttf)
New Tai Lue	https://cdn.flexmonster.com/fonts/NotoSansNewTaiLue-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansNewTaiLue-Regular.ttf)
Ogham	https://cdn.flexmonster.com/fonts/NotoSansOgham-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansOgham-Regular.ttf)
Ol Chiki	https://cdn.flexmonster.com/fonts/NotoSansOlChiki-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansOlChiki-Regular.ttf)
Old Italic	https://cdn.flexmonster.com/fonts/NotoSansOldItalic-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansOldItalic-Regular.ttf)
Old Persian	https://cdn.flexmonster.com/fonts/NotoSansOldPersian-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansOldPersian-Regular.ttf)
Old South Arabian	https://cdn.flexmonster.com/fonts/NotoSansOldSouthArabian-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoS

	ansOldSouthArabian-Regular.ttf)
Old Turkic	https://cdn.flexmonster.com/fonts/NotoSansOldTurkic-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansOldTurkic-Regular.ttf)
Oriya	https://cdn.flexmonster.com/fonts/NotoSansOriya-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansOriya-Regular.ttf)
Oriya UI	https://cdn.flexmonster.com/fonts/NotoSansOriyaUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansOriyaUI-Regular.ttf)
Osage	https://cdn.flexmonster.com/fonts/NotoSansOsage-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansOsage-Regular.ttf)
Osmanya	https://cdn.flexmonster.com/fonts/NotoSansOsmanya-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansOsmanya-Regular.ttf)
Phags-pa	https://cdn.flexmonster.com/fonts/NotoSansPhagsPa-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansPhagsPa-Regular.ttf)
Phoenician	https://cdn.flexmonster.com/fonts/NotoSansPhoenician-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansPhoenician-Regular.ttf)
Rejang	https://cdn.flexmonster.com/fonts/NotoSansRejang-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansRejang-Regular.ttf)
Runic	https://cdn.flexmonster.com/fonts/NotoSansRunic-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansRunic-Regular.ttf)
Samaritan	https://cdn.flexmonster.com/fonts/NotoSansSamaritan-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansSamaritan-Regular.ttf)
Saurashtra	https://cdn.flexmonster.com/fonts/NotoSansSaurashtra-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansSaurashtra-Regular.ttf)
Shavian	https://cdn.flexmonster.com/fonts/NotoSansShavian-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansShavian-Regular.ttf)
Sinhala	https://cdn.flexmonster.com/fonts/NotoSansSinhala-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansSinhala-Regular.ttf)
Sinhala UI	https://cdn.flexmonster.com/fonts/NotoSansSinhalaUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansSinhalaUI-Regular.ttf)
Sundanese	https://cdn.flexmonster.com/fonts/NotoSansSundanese-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansSundanese-Regular.ttf)
Syloti Nagri	https://cdn.flexmonster.com/fonts/NotoSansSylotiNagri-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansSylotiNagri-Regular.ttf)
Symbols	https://cdn.flexmonster.com/fonts/NotoSansSymbols-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansSymbols-Regular.ttf)
Symbols2	https://cdn.flexmonster.com/fonts/NotoSansSymbols2-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansSymbols2-Regular.ttf)

Syriac Eastern	https://cdn.flexmonster.com/fonts/NotoSansSyriacEastern-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansSyriacEastern-Regular.ttf)
Syriac Estrangela	https://cdn.flexmonster.com/fonts/NotoSansSyriacEstrangela-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansSyriacEstrangela-Regular.ttf)
Syriac Western	https://cdn.flexmonster.com/fonts/NotoSansSyriacWestern-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansSyriacWestern-Regular.ttf)
Tagalog	https://cdn.flexmonster.com/fonts/NotoSansTagalog-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansTagalog-Regular.ttf)
Tagbanwa	https://cdn.flexmonster.com/fonts/NotoSansTagbanwa-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansTagbanwa-Regular.ttf)
Tai Le	https://cdn.flexmonster.com/fonts/NotoSansTaiLe-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansTaiLe-Regular.ttf)
Tai Tham	https://cdn.flexmonster.com/fonts/NotoSansTaiTham-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansTaiTham-Regular.ttf)
Tai Viet	https://cdn.flexmonster.com/fonts/NotoSansTaiViet-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansTaiViet-Regular.ttf)
Tamil	https://cdn.flexmonster.com/fonts/NotoSansTamil-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansTamil-Regular.ttf)
Tamil UI	https://cdn.flexmonster.com/fonts/NotoSansTamilUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansTamilUI-Regular.ttf)
Telugu	https://cdn.flexmonster.com/fonts/NotoSansTelugu-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansTelugu-Regular.ttf)
Telugu UI	https://cdn.flexmonster.com/fonts/NotoSansTeluguUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansTeluguUI-Regular.ttf)
Thaana	https://cdn.flexmonster.com/fonts/NotoSansThaana-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansThaana-Regular.ttf)
Thai	https://cdn.flexmonster.com/fonts/NotoSansThai-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansThai-Regular.ttf)
Thai UI	https://cdn.flexmonster.com/fonts/NotoSansThaiUI-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansThaiUI-Regular.ttf)
Tibetan	https://cdn.flexmonster.com/fonts/NotoSansTibetan-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansTibetan-Regular.ttf)
Tifinagh	https://cdn.flexmonster.com/fonts/NotoSansTifinagh-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansTifinagh-Regular.ttf)
Ugaritic	https://cdn.flexmonster.com/fonts/NotoSansUgaritic-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansUgaritic-Regular.ttf)
Vai	https://cdn.flexmonster.com/fonts/NotoSansVai-

	Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansVai-Regular.ttf)
Yi	https://cdn.flexmonster.com/fonts/NotoSansYi-Regular.ttf (https://cdn.flexmonster.com/fonts/NotoSansYi-Regular.ttf)

How to export to the server without using a browser

You can easily export the report without a browser using Puppeteer (https://pptr.dev/) — a JavaScript library for working with headless browsers.

We prepared a sample GitHub project (https://github.com/flexmonster/pivot-puppeteer) with Flexmonster and Puppeteer. It demonstrates how to export Flexmonster reports in headless browsers with Puppeteer. To run the sample project, follow these steps:

1. Download the .zip archive with the sample or clone it from GitHub (https://github.com/flexmonster/pivot-puppeteer) with the following command:

```
git clone https://github.com/flexmonster/pivot-puppeteer
cd pivot-puppeteer
```

2. Install the npm packages described in package.json:

```
npm install
```

3. Run the project:

```
npm start
```

When the export is complete, find the saved files in the storage/ folder.

Now let’s have a look at the project files’ contents to understand how the sample project works:

index.html

The index.html file (https://github.com/flexmonster/pivot-puppeteer/blob/master/index.html) contains the pivot table subscribed to the ready, reportcomplete, and exportcomplete events. These events will let us know when the report is ready to be exported and when the export is finished.

When the component triggers one of these events, the dispatchEvent() (https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/dispatchEvent) method triggers the same event for the browser window. This approach allows the browser to handle the component’s events in its scope.

You can specify other Flexmonster events (https://www.flexmonster.com/api/events/) similarly.

pivot.js

The pivot.js file (https://github.com/flexmonster/pivot-puppeteer/blob/master/pivot.js) runs the browser and performs the export. This section describes its key points.

In the sample project, we use ready, reportcomplete, and exportcomplete events to export the report. You can add other Flexmonster events in a similar way – check it out (https://github.com/flexmonster/pivot-puppeteer/blob/master/pivot.js#L121-L140).

The next important part of the project is where we set a report. It’s done using the setReport() (https://github.com/flexmonster/pivot-puppeteer/blob/master/pivot.js#L68-L72) function as soon as the component is initialized. You can replace the default report with an inline report or a link to it.

The exportTo() function supports changing the file name or exporting the report to your server – just specify the corresponding export parameters. The structure of the parameters is the same as in the flexmonster.exportTo() API call (https://www.flexmonster.com/api/exportto/).

For technical details on how the export is performed, see comments in the pivot.js file.

6.13. Calculated values

Calculated values give you the option to add measures that were missing in the original data. They can be saved and restored within the report. This feature is available for "json", "csv" and "api" data source types. Each calculated measure is described inside a measure object. It can have the following parameters:

- uniqueName – String. The measure’s unique name. This property will be used as an identifier for the measure inside Flexmonster and as an identifier to remove the measure via API.
- formula – String. Represents the formula. It can contain the following operators:+, -, *, / and the following functions:isNaN(), !isNaN() (check a full list (#formula-operators)). Other measures can be addressed using the measure’s unique name and aggregation function, for example sum("Price") or max("Order"). To see the list of supported aggregation functions for each data source type, refer to Flexmonster’s technical specifications (/technical-specifications/#aggregations).
- caption (optional) – String. The measure’s caption.
- grandTotalCaption (optional) – String. The measure’s grand total caption.
- active (optional) – Boolean. Indicates whether the measure will be selected for the report (true) or not (false). active: false can be useful if the measure has non-default properties, but should not be selected for the grid or the chart.
- individual (optional) – Boolean. Only for "csv" and "json" data source types. Defines whether the formula is calculated using raw values (true) or using aggregated values (false). *Default value: false*.
- calculateNaN (optional) – Boolean. Defines whether the formula is calculated using NaN values (true) or using null values (false). *Default value: true*.
- format (optional) – String. The name of the number formatting that will be applied to the measure. Measure values can be formatted according to the number formatting defined in the report. All available number formattings are stored in the formats array in the report. More information about the number formatting part of the report can be found in the number formatting (https://www.flexmonster.com/doc/number-formatting/) article.

This example on JSFiddle (https://jsfiddle.net/flexmonster/x3qw9s71/) illustrates how to define a calculated measure with the minimum price for each color. The slice is defined like this:

```
slice: {
 rows: [
  { uniqueName: "Color" }
 ],
 measures: [
  { uniqueName: "Price", aggregation: "sum"},
  {
   formula: 'min("Price")',
   uniqueName: "Min Price",
   caption: "Min Price",
   active: true
  }
 ]
}
```

The next example (https://jsfiddle.net/flexmonster/x0pe8azg/) illustrates how to define a calculated measure with a more complex formula. To highlight the values you can add conditional formatting for the Top Category measure:

```
slice: {
 rows: [
  { uniqueName: "Color" }
 ],
 measures: [
  { uniqueName: "Price", aggregation: "sum"},
  {
   uniqueName: "Top Category",
   formula: 'average("Price") < 4000 and sum("Quantity") > 100',
   caption: "Top Category",
   active: true
  }
 ]
},
conditions: [
    {
        formula: "#value = 1",
        measure: "Top Category",
        format: {
            backgroundColor: "#66FF99",
            color: "#000000",
            fontFamily: "Arial",
            fontSize: "12px"
        }
    }
]
```

This JSFiddle (https://jsfiddle.net/flexmonster/qkhe4wkq/) example shows how to specify the number formatting for your calculated measure.

The individual property allows the formula to be calculated using raw values. In the example (https://jsfiddle.net/flexmonster/7mtwxqow/) the formula sum('Price') * sum('Amount') will be calculated like this:
set individual: true: 174 * 36 + 225 * 44
set individual: false: (174 + 225) * (36 + 44)

The following report illustrates how to use the individual property:

```
{
 dataSource: {
  data: [
         {
             "Country" : "Canada",
             "Amount" : 36,
             "Price" : 174
         },
         {
             "Country" : "Canada",
             "Amount" : 44,
             "Price" : 225
         }
     ]
 },
 slice: {
  rows: [
      {
          uniqueName: "Country"
      }
  ],
  measures: [
      {
          uniqueName: "Price",
          aggregation: "sum",
          active: true
      },
      {
          uniqueName: "Overall price",
          formula: "sum('Price') * sum('Amount')",
          individual: true,
          caption: "Overall price",
          active: true
      }
  ]
 }
}
```

For more examples of adding calculated values, see the Examples page (https://www.flexmonster.com/examples/#!calculated-measures).

A full list of operators and functions for calculated values

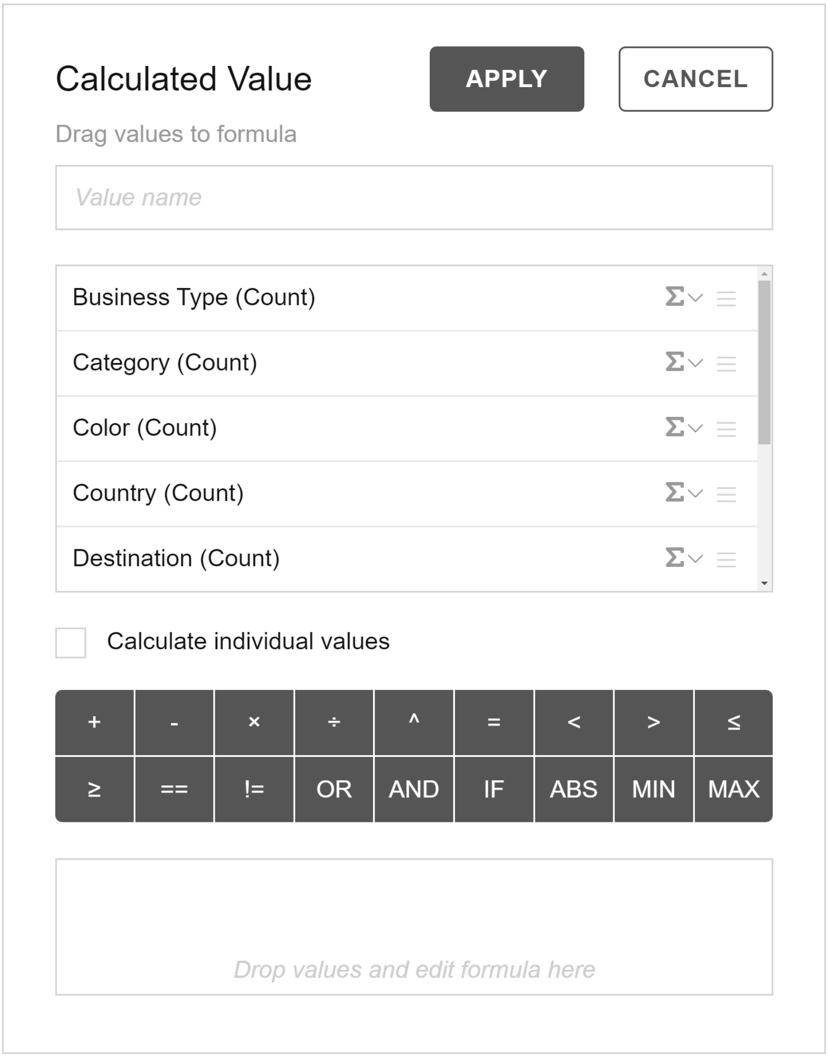
Below is a list of all operators and functions supported in formula:

- + – arithmetic addition operator. Syntax: a + b.
- - – arithmetic subtraction operator. Syntax: a - b.
- * – arithmetic multiplication operator. Syntax: a * b.
- / – arithmetic division operator. Syntax: a / b.
- ^ – arithmetic power operator. Syntax: a^2.

- < – comparison less than operator. Syntax: a < b.
- <= – comparison less than or equal operator. Syntax: a <= b.
- > – comparison greater than operator. Syntax: a > b.
- >= – comparison greater than or equal operator. Syntax: a >= b.
- == – comparison equal operator. Syntax: a == b.
- != – comparison not equal operator. Syntax: a != b.
- or – logical OR operator. Syntax: a or b.
- and – logical AND operator. Syntax: a and b.
- if – conditional operator. Syntax: if(condition, then, else).
- abs – function that returns the absolute value of a number. Syntax: abs(number).
- min – function that returns the minimum value. Syntax: min(number1, number2).
- max – function that returns the maximum value. Syntax: max(number1, number2).
- isNaN – function that checks whether the value is not a number. Syntax: isNaN(value).
- !isNaN – function that checks whether the value is a number. Syntax: !isNaN(value).

Add calculated values using the Field List

Use Add calculated value in the Field List to add the calculated measure at runtime.



Calculated values via API

Calculated measures can be defined within the report or added via the addCalculatedMeasure()

(/api/addcalculatedmeasure/) API call. To remove a calculated measure use the removeCalculatedMeasure() (/api/removecalculatedmeasure/) API call. removeAllCalculatedMeasures() (/api/removeallcalculatedmeasures/) removes all calculated measures.

6.14. Custom sorting

Flexmonster Pivot allows you to set the specific sorting order for hierarchy members in columns, rows, or report filters. By default, hierarchy members are sorted alphabetically in the component. In OLAP cubes, Flexmonster takes the order that was defined inside the cube. The following options are available to define sorting:

1. Set alphabetical, reverse alphabetical, or an unsorted order for members (#set-order).
2. Define custom sorting using a sorting function (#define-sorting-function).
3. Define custom sorting using an array (for JSON and CSV) (#define-sorting-in-array).

Set alphabetical, reverse alphabetical, or an unsorted order for members

Alphabetical order is used by default. To set reverse alphabetical order or display members unsorted, use setSort() (/api/setsort/). This API call has the following parameters:

- hierarchyName – String. The name of the hierarchy to which the sorting is applied.
- sortType – String. The following sorting types can be applied: "asc", "desc", or "unsorted".

Here is a simple example:

```
flexmonster.setSort("Category", "desc");
```

Try the example on JSFiddle (https://jsfiddle.net/flexmonster/9h15aeye/).

The sorting type can also be specified directly in the report:

```
{
    "dataSource": {
        "filename": "https://www.flexmonster.com/download/data.csv"
    },
    "slice": {
        "rows": [
            {
                "uniqueName": "Category",
                "sort": "desc"
            }
        ],
        "measures": [ {"uniqueName": "Price"} ]
    }
}
```

Define custom sorting using a sorting function

The sort order can be set via sortingMethod() (/api/sortingmethod/). This API call has the following parameters:

- hierarchyName – String. The unique name of the hierarchy to which the sorting is applied.

- compareFunction – Function. Defines the sort order. The input parameters are the same as for the compareFunction of the Array.sort() method (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort).

Here is a simple example, where the new sorting function is applied to the "Category" hierarchy:

```
flexmonster.sortingMethod("Category", function (a, b) {
    return a < b ? 1 : -1;
});
```

Try the example on JSFiddle (https://jsfiddle.net/flexmonster/j85qom2p/).

Adding a custom sorting function is used to override the default alphabetical order. For example, you can make sure that some members are always kept at the end or at the start of the list.

Define custom sorting using an array (for JSON and CSV)

Custom sorting for JSON and CSV data sources can be set via the sortOrder array. This property can be specified like so: ["member_1", "member_2", etc.].

We will analyze a simple example to understand how it works. There is a report with one hierarchy defined as a row:

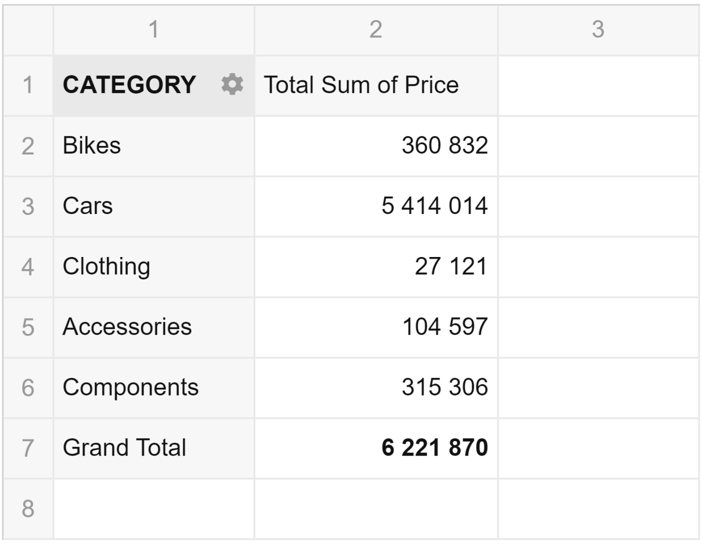
```
{
    "dataSource": {
        "filename": "http://www.flexmonster.com/download/data.csv"
    },
    "slice": {
        "rows": [
            {
                "uniqueName": "Category"
            }
        ],
        "measures": [ {"uniqueName": "Price"} ]
    }
}
```

This hierarchy has the following members: “Accessories”, “Bikes”, “Clothing”, “Components”, and “Cars”. To define custom sorting we add the sortOrder property:

```
{
    "dataSource": {
        "filename": "http://www.flexmonster.com/download/data.csv"
    },
    "slice": {
        "rows": [
            {
                "uniqueName": "Category",
                "sortOrder": ["Bikes", "Cars", "Clothing",
                     "Accessories", "Components"]
            }
        ],
        "measures": [ {"uniqueName": "Price"} ]
    }
}
```

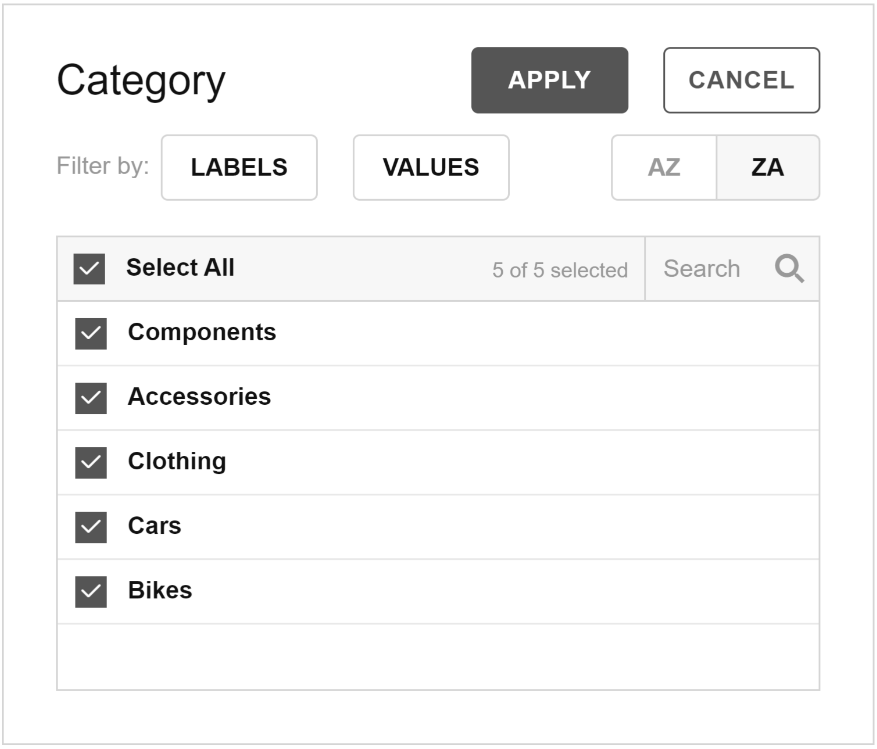
Open the example on JSFiddle (https://jsfiddle.net/flexmonster/39z1xy8y/).

Now hierarchy members will be displayed in the predefined order:



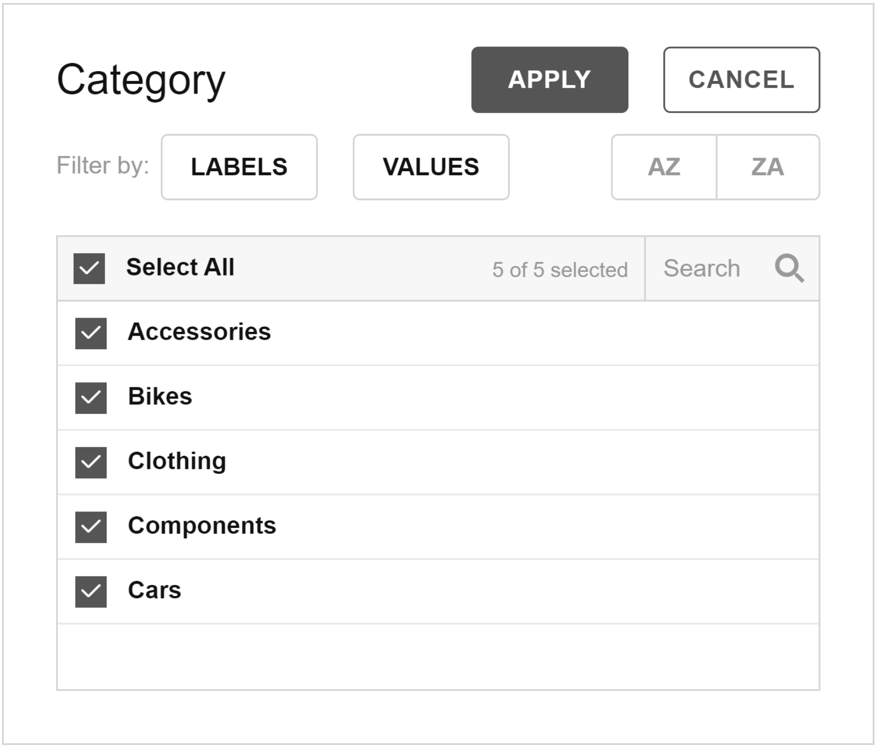
	1	2	3
1	CATEGORY	Total Sum of Price	
2	Bikes	360 832	
3	Cars	5 414 014	
4	Clothing	27 121	
5	Accessories	104 597	
6	Components	315 306	
7	Grand Total	**6 221 870**	
8			

The reverse alphabetical order will sort in the opposite order to the one defined in the sortOrder property:



- ✓ Components
- ✓ Accessories
- ✓ Clothing
- ✓ Cars
- ✓ Bikes

If you remove both alphabetical and reverse alphabetical sorting, hierarchy members will be displayed in the same order they came from the data source:



Custom sorting is an easy way to predefine your own order for columns, rows, or report filters in the slice.

7. Charts

7.1. Available tutorials

This section lists the available guides for working with Flexmonster Pivot Charts and 3rd party charting libraries.

Flexmonster Pivot Charts are the component's built-in charts. The charts are easy to use and configure as they have many options and API calls. The guide on Flexmonster Pivot Charts (/doc/flexmonster-pivot-charts/) describes how to use them as well as the available chart types and options.

3rd party charting library integration

To extend Flexmonster's charting functionality, see the ready-to-use integrations with 3rd party charting libraries:

- Highcharts (/doc/integration-with-highcharts/)
- amCharts (/doc/integration-with-amcharts/)
- Google Charts (/doc/integration-with-google-charts/)

- FusionCharts (/doc/integration-with-fusioncharts/)

Integrating Flexmonster with other charting libraries is also possible – see Integration with any charting library (/doc/integration-with-any-charting-library/).

7.2. Flexmonster Pivot Charts

Our component has built-in interactive charts that extend its visualization functionality. Flexmonster Pivot Charts allow expanding hierarchies, filtering data, and drilling through chart segments for more details about the data.

This tutorial has the following sections to help you get started with Flexmonster Pivot Charts:

- Available chart types (#available-charts)
- Which data is shown on the pivot charts (#slice-on-charts)
- Using Flexmonster Pivot Charts (#using-pivot-charts)
- Configuring the pivot charts (#configuring-pivot-charts)
- Customizing the pivot charts (#customizing-pivot-charts)
- How the pivot charts are created (#how-pivot-charts-are-created)

Available chart types

Flexmonster provides seven chart types:

- Column
- Bar
- Pie
- Line
- Scatter
- Column line
- Stacked column

You can try all the available chart types on the live Pivot Charts demo (/demos/pivot-charts/).

If you need a chart type that is not in the list, we recommend using one of these 3rd party charting libraries:

- Highcharts (/doc/integration-with-highcharts/)
- amCharts (/doc/integration-with-amcharts/)
- FusionCharts (/doc/integration-with-fusioncharts/)
- Google charts (/doc/integration-with-google-charts/)

To integrate Flexmonster with other charting libraries, see our guide on integrating with any charting library (/doc/integration-with-any-charting-library/).

Which data is shown on the pivot charts

Flexmonster Pivot Charts visualize the data subset defined in report.slice. To show a different data subset, change the slice either via the UI using the Field List, or programmatically with the report.slice object.

To learn more about slice configuration, refer to the guide about the slice (/doc/slice/).

Using Flexmonster Pivot Charts

Create a basic pivot table

Add the following pivot table to your HTML page using data from a CSV file:

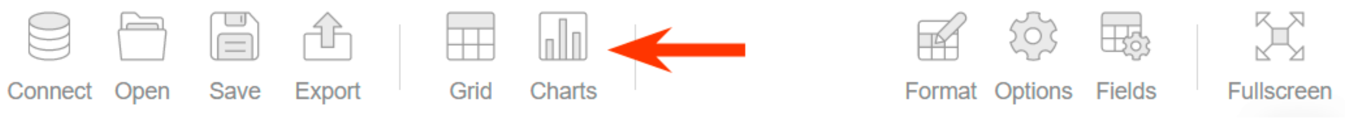
```
<div id="pivotContainer">The component will appear here</div>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>

<script>
    var pivot = new Flexmonster({
        container: "pivotContainer",
        componentFolder: "https://cdn.flexmonster.com/"
        toolbar: true,
        report: {
            dataSource: {
                filename: "data.csv"
            }
        }
    });
</script>
```

The grid is the default view shown by the component. Switching to charts can be done either using the Toolbar or programmatically.

Switch to the pivot charts via the Toolbar

This is the simplest way to switch to the pivot charts. Select the Charts icon on the Toolbar to see the pivot charts:



Switch to the pivot charts programmatically

The default view can be changed from the grid to the charts by setting the viewType option to "charts":

```
report: {
    dataSource: {
        filename: "data.csv"
    },
    options: {
        viewType: "charts"
    }
},
```

See the guide on options (/doc/options/) to learn more.

Configuring the pivot charts

Chart options

Options specific to charts can be set via the options.charts property. It lets you manage the chart type, chart title, visibility of the UI controls, etc.

Here is the example of how the chart title can be set:

```
options: {
    viewType: "charts",
    charts: {
        title: "Summary chart"
    }
}
```

Try a live sample on JSFiddle (https://jsfiddle.net/flexmonster/bmvhd7bt/).

See the full list of chart options (/doc/options/#chart).

General options

General options allow configuring filters, the Field List button, and many more. They are applied to both the grid and the charts. Here is the example that shows how to hide the Field List button:

```
options: {
    viewType: "charts",
    configuratorButton: false
}
```

Check out a live demo on JSFiddle (https://jsfiddle.net/flexmonster/fky93q1f/).

See the full list of general options (/doc/options/#general-options).

Customizing the pivot charts

The pivot charts' appearance can be customized and adjusted to the user's needs. See our detailed guide (/doc/customizing-pivot-charts/) on chart customizing.

How the pivot charts are created

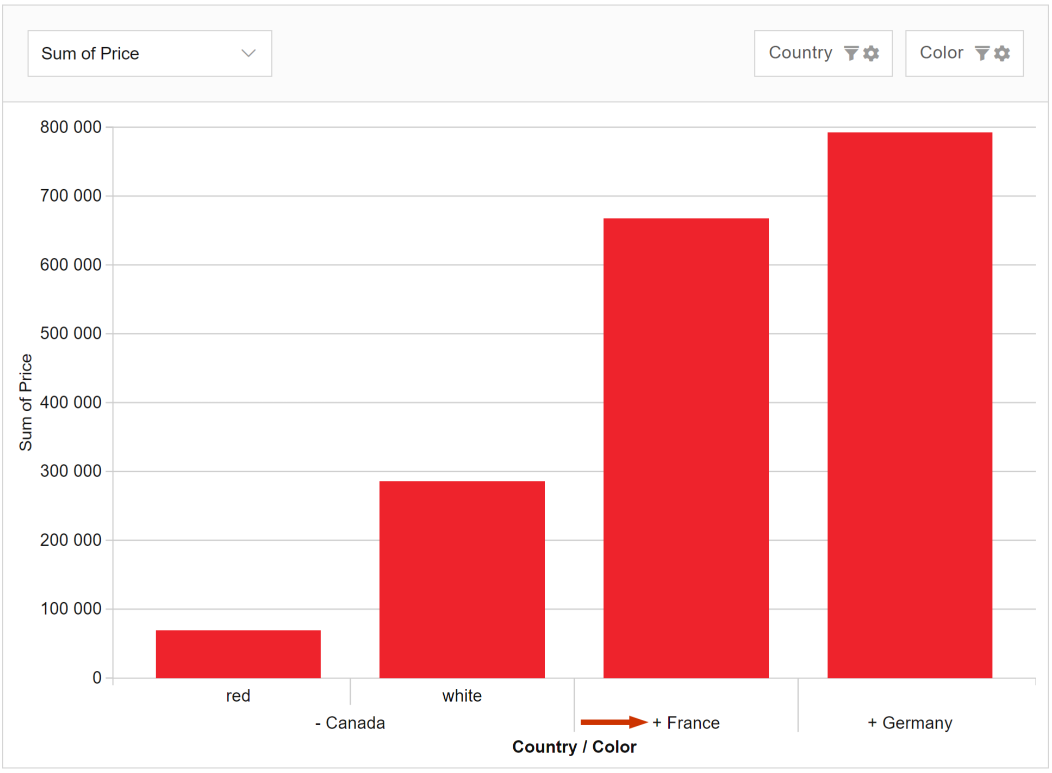
This section explains how different types of Flexmonster Pivot Charts are created.

XY chart

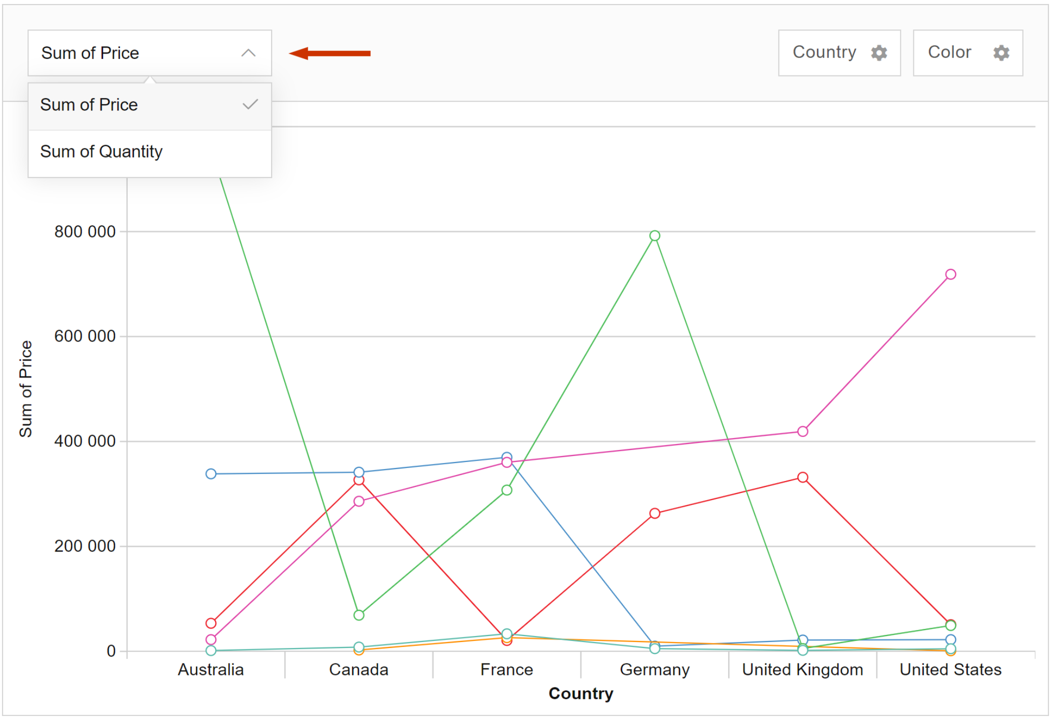
XY is a general title that implies many chart types with similar features. Flexmonster Pivot Charts suggest six types of XY chart: column, bar, line, scatter, stacked column, and column line charts. This section focuses on similarities in their structure.

XY chart is called so due to the X-axis and Y-axis used in diagrams of this type. Measures go to the Y-axis (vertical), and row fields go to the X-axis (horizontal). Therefore, one measure and one field in rows are enough for a basic XY chart.

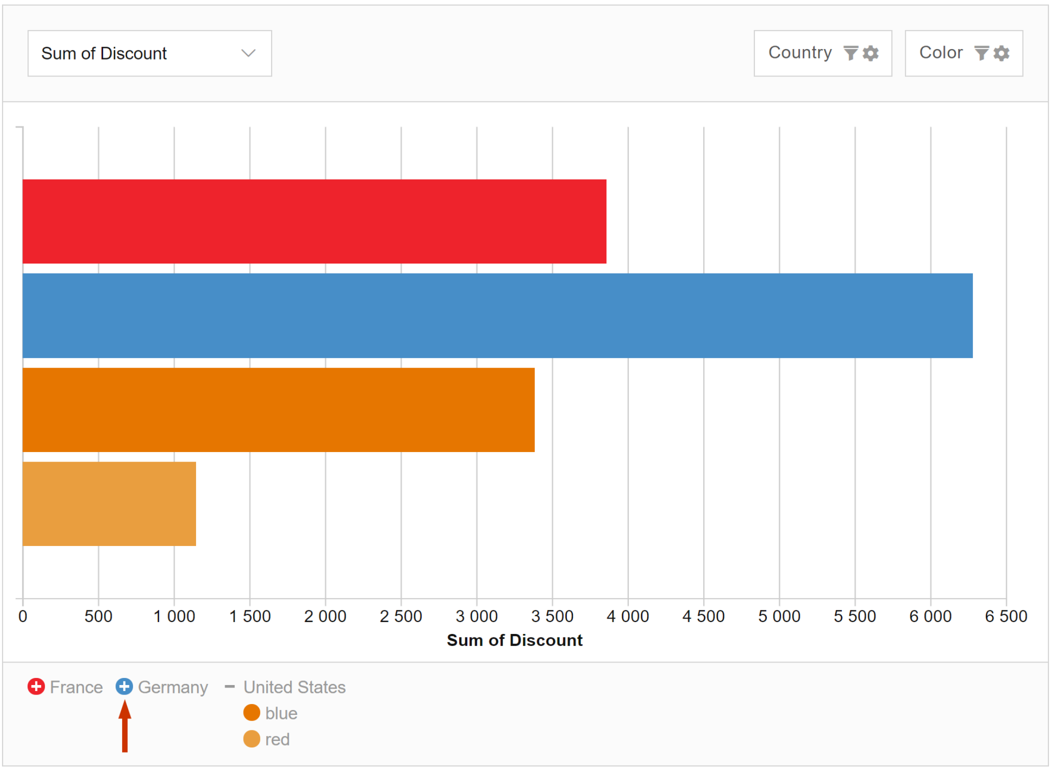
If rows contain more than one field, you can expand members for more details on the data. Just click a + icon near the member name:



If the slice contains two or more measures, you can choose a chart measure via the options.chart.activeMeasure (https://www.flexmonster.com/api/options-object/#activeMeasure) property or via UI:



When columns are present in the slice, the legend appears below the chart. If columns have more than one field, you can expand a chart segment by clicking a + icon on the legend:



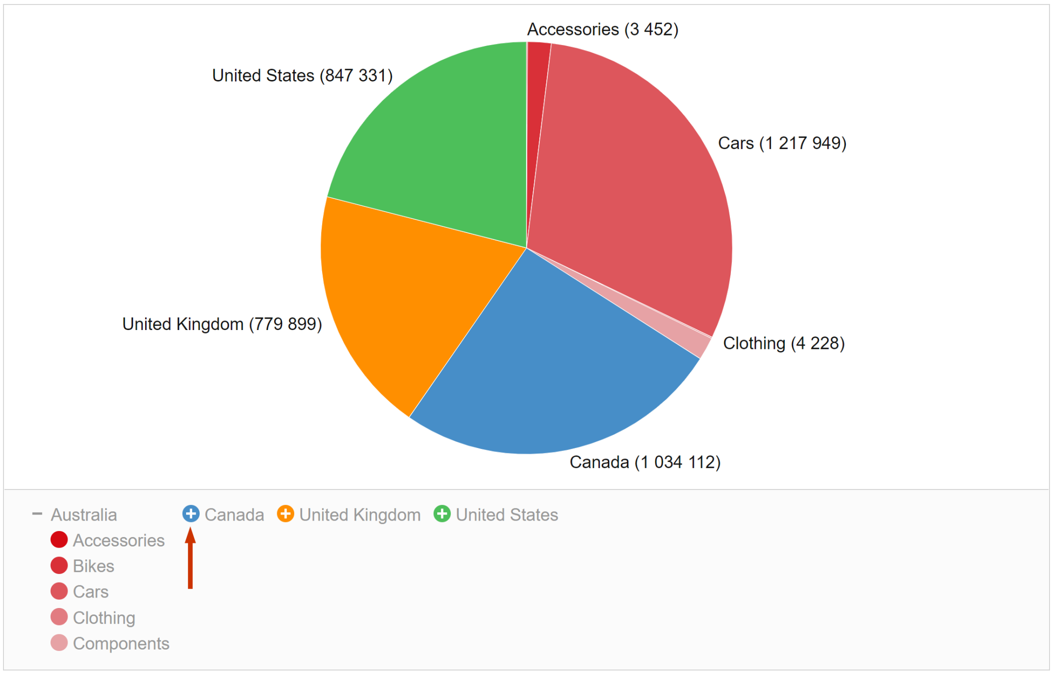
Most of the Flexmonster chart types have a similar structure, but the column line chart is built differently. This type of chart uses two measures: the first measure is a base for a column chart, and the second measure is used for a line chart.

The options.chart.activeMeasure (https://www.flexmonster.com/api/options-object/#activeMeasure) property isn't supported for the column line chart. Active measures are the first two measures from the slice.

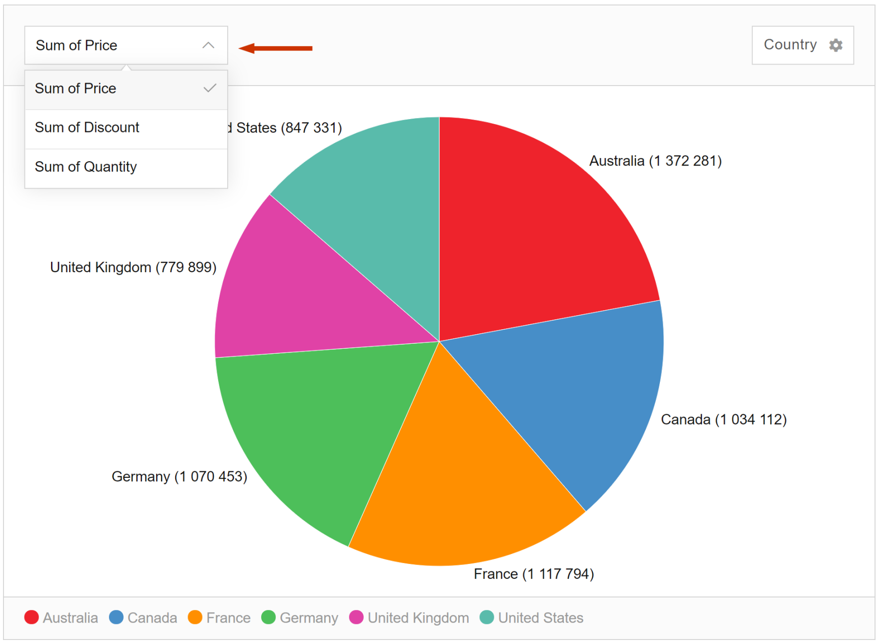
Pie chart

For a basic pie chart, it is enough to have one field in rows and one measure. The field will be a pie chart's category, and the measure will be the chart's numeric value.

When rows contain more than one field, you can expand members to get extra details on the data. Just click a + icon on the chart legend to expand a chart segment:

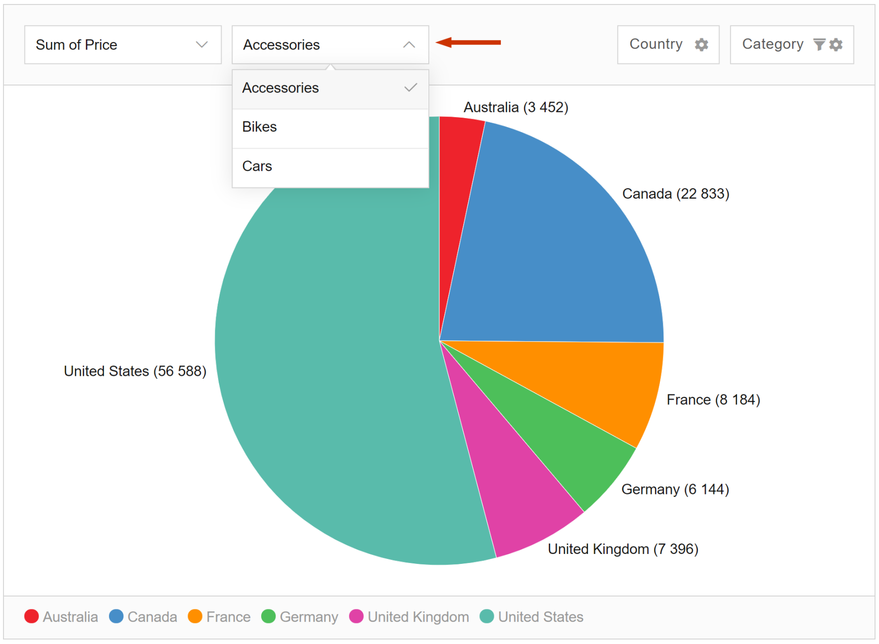


If the slice contains two or more measures, you can choose a chart measure via the options.chart.activeMeasure (/api/options-object/#activeMeasure) property or via UI:



When columns are defined in the slice, they are also used for the chart. By default, a pie chart is based on the data from the first member of the first column field. Sorting is considered as well.

You can select a member for the chart via the options.chart.pieDataIndex (/api/options-object/#pieDataIndex) property or via UI:



If columns have more than one field, the fields are used for the chart as follows:

- If all the column nodes are collapsed, only the first column field is used for a pie chart.
- If some nodes are expanded, then sub-members of expanded nodes can be selected for the chart.

What’s next?

You may be interested in the following articles:

- Integration with Highcharts (/doc/integration-with-highcharts/)
- Integration with amCharts (https://www.flexmonster.com/doc/integration-with-amcharts/)
- Integration with FusionCharts (/doc/integration-with-fusioncharts/)
- Integration with Google Charts (/doc/integration-with-google-charts/)

7.3. Integration with Highcharts

This tutorial will help you integrate Flexmonster with the Highcharts (https://www.highcharts.com/) charting library.

The guide contains the following sections:

- Supported chart types (#supported-charts)
- Adding Highcharts (#adding-highcharts)
- Flexmonster Connector for Highcharts (#highcharts-connector)

Supported chart types

Flexmonster supports the following Highchart types:

- area (demo (https://jsfiddle.net/flexmonster/tuhnwwcn/))
- arearange (demo (https://jsfiddle.net/flexmonster/z21f87y3/))
- areaspline (demo (https://jsfiddle.net/flexmonster/wmdd9970/))
- areasplinerange (demo (https://jsfiddle.net/flexmonster/atL2uh9k/))
- bar (demo (https://jsfiddle.net/flexmonster/qf73smhf/))

- bubble (demo (https://jsfiddle.net/flexmonster/qLxdwu5x/))
- column (demo (https://jsfiddle.net/flexmonster/L89wqzj9/))
- columnrange (demo (https://jsfiddle.net/flexmonster/6zrd63r0/))
- errorbar (demo (https://jsfiddle.net/flexmonster/da8mpt9a/))
- funnel (demo (https://jsfiddle.net/flexmonster/kh1eL9a6/))
- line (demo (https://jsfiddle.net/flexmonster/2xoqujba/))
- pie (demo (https://jsfiddle.net/flexmonster/j139y3xe/))
- polygon (demo (https://jsfiddle.net/flexmonster/0bpt20s8/))
- pyramid (demo (https://jsfiddle.net/flexmonster/q4w37pya/))
- scatter (demo (https://jsfiddle.net/flexmonster/gdnxwaj1/))
- spline (demo (https://jsfiddle.net/flexmonster/9ay5ugbr/))
- waterfall (demo (https://jsfiddle.net/flexmonster/bvh3f4pn/))

If the chart type that you want to use is not on the list, it is possible to use prepareDataFunction to preprocess the data to fit your preferences.

More ready-to-use examples of integration with Highcharts you can find on the Examples page (https://www.flexmonster.com/examples/#!highcharts).

Adding Highcharts

1. Add the component using data from a CSV file to your HTML page. Replace XXXX-XXXX-XXXX-XXXX-XXXX with your license key. If you don’t have a license key, contact our team (https://www.flexmonster.com/contact/) and request a special trial key.

```
<div id="pivotContainer">The component will appear here</div>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>

<script>
    var pivot = new Flexmonster({
        container: "pivotContainer",
        componentFolder: "https://cdn.flexmonster.com/"
        toolbar: true,
        report: {
            dataSource: {
                filename: "data.csv"
            },
            slice: {
                rows: [
                    { uniqueName: "Country" }
                ],
                columns: [
                    { uniqueName: "Business Type" },
                    { uniqueName: "[Measures]" }
                ],
                measures: [
                    { uniqueName: "Price" }
                ]
            }
        },
        licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
    });
</script>
```

2. Add Highcharts:

```
<script src="https://code.highcharts.com/highcharts.js"></script>
```

3. Add our connector for Highcharts:

```
<script src="https://cdn.flexmonster.com/lib/flexmonster.highcharts.js"></scri
pt>
```

4. Add a container for the chart:

```
<div id="highchartsContainer"></div>
```

5. Add a reportComplete event handler to know when the pivot table is ready to be a data provider:

```
reportcomplete: function() {
    pivot.off("reportcomplete");
    createChart();
}
```

6. Add a function to create the chart. This function uses the connector for Highcharts from flexmonster.highcharts.js. The connector extends the Flexmonster API with the following call: highcharts.getData(options, callbackHandler, updateHandler).

```
function createChart() {
    pivot.highcharts.getData(
        {},
        function(chartConfig) {
            Highcharts.chart('highchartsContainer', chartConfig);
        },
        function(chartConfig) {
            Hightcharts.chart('highchartsContainer', chartConfig);
        }
    );
}
```

You will see a line chart that displays the same data that is shown in the pivot pivot table instance and it will react to updates. Check out a live example on JSFiddle (https://jsfiddle.net/flexmonster/2xoqujba/).

Flexmonster Connector for Highcharts

flexmonster.highcharts.js is a connector between our component and Highcharts. The highcharts.getData() method requests data from the pivot table and preprocesses it to the appropriate format for the required type of chart.

highcharts.getData(options:Object, callbackHandler:Function, updateHandler:Function) has the following parameters:

- options – Object. This object has the following properties:
    - type (optional) – String. The chart type to prepare data for. *Default value:* "line".

      ```
      function pie() {
          pivot.highcharts.getData({
              type: 'pie'
          }, function(chartConfig) {
              Highcharts.chart('highchartsContainer', chartConfig);
      ```

```
    }, function(chartConfig) {
        Highcharts.chart('highchartsContainer', chartConfig);
    });
}
```

    - slice (optional) – Object. Defines the slice from which the data should be returned (for JSON and CSV data sources only). If not defined, the API call will return data currently displayed in the pivot table.

      ```
      function withManyYAxis() {
          pivot.highcharts.getData({
              type: 'column',
              slice: {
                  rows: [{uniqueName: "Country"}],
                  columns: [{uniqueName: "[Measures]"}],
                  measures: [
                      {uniqueName: "Price"},
                      {uniqueName: "Quantity"}
                  ]
              }
          }, function(chartConfig) {
              Highcharts.chart('highchartsContainer', chartConfig);
          }, function(chartConfig) {
              Highcharts.chart('highchartsContainer', chartConfig);
          });
      }
      ```

    - xAxisType (optional) – String. Set the value of this to “datetime” to use dates on the X-axis. Try it on JSFiddle (https://jsfiddle.net/flexmonster/pzo5meke/), this sample also illustrates how to define a type for particular series.
    - valuesOnly (optional) – Boolean. Set this property to true to display all axes values in the chart as numbers. This property is applicable only for the following chart types: “bubble”, “line”, “polygon”, and “spline”. Note, that this value is always true for the “scatter” chart (and you do not need to set it explicitly to true). Try it on JSFiddle (https://jsfiddle.net/flexmonster/wrt82p00/). *Default value:* false.
    - withDrilldown (optional) – Boolean. Set this property to true to have drill-downs in the chart. This property is available for the following chart types: “area”, “areaspline”, “bar”, “column”, “waterfall”, “funnel”, “pie”, “pyramid”, “polygon”, “spline”, and “line”. Try it on JSFiddle (https://jsfiddle.net/flexmonster/qf73smhf/). *Default value:* false.
    - prepareDataFunction (optional) – An external function. If the connector does not include the necessary type of chart or if you need to preprocess the data differently, use this function. prepareDataFunction takes two input parameters: rawData – the raw data (check out the structure of rawData in getData() (52.207.238.113/api/getdata/)); options – an object with options set in the highcharts.getData() function. Try it on JSFiddle (https://jsfiddle.net/flexmonster/x0cqafqh/).
- callbackHandler – Function. Specifies what will happen once the data is ready. Additional options can be specified and then data can be passed directly to the charting library. Takes two input parameters – chartConfig and rawData (rawData is passed just in case you need it, for example for defining number formatting in the tooltip).
- updateHandler (optional) – Function. Takes two input parameters – chartConfig and rawData. Specifies what will happen once data in the pivot table is filtered/sorted/etc or a number format is changed.

The connector has several functions for defining number formatting for Highcharts. All these functions take the pivot table format object and return the formatting string in Highcharts format. The Highcharts format string is a template for labels with one of the following variables inserted in a bracket notation: value, point.x, point.y, or point.z. For example, "{value:.2f}" is for two decimal places. Thus, there are four functions in the connector that convert the Flexmonster number format object to the Highcharts format string (refer to the Highcharts documentation for more details about the format strings).

- highcharts.getAxisFormat(format) – String. Takes the pivot table format object and returns the Highcharts format string with the value variable.
- highcharts.getPointXFormat(format) – String. Takes the pivot table format object and returns the Highcharts format string with the point.x variable.
- highcharts.getPointYFormat(format) – String. Takes the pivot table format object and returns the Highcharts format string with the point.y variable.
- highcharts.getPointZFormat(format) – String. Takes the pivot table format object and returns the Highcharts format string with the point.z variable.

These functions can be used in both callbackHandler and updateHandler to define the number formatting for axes and tooltips. Try it on JSFiddle (https://jsfiddle.net/flexmonster/z0bod0w9/).

7.4. Integration with amCharts

This guide describes how to integrate Flexmonster with amCharts (https://www.amcharts.com/) – a JavaScript library for interactive data visualization.

The approach described in the tutorial works only for amCharts 4 (https://www.amcharts.com/docs/v4/). To integrate Flexmonster with older amCharts versions, follow this guide: Integration with any charting library (https://www.flexmonster.com/doc/integration-with-any-charting-library/).

See the sections below to integrate Flexmonster with the amCharts library:

- Supported chart types (#supported-charts)
- Adding amCharts (#adding-amcharts)
- Preparing data for the chart (#data-preparation)
- Configuring number formatting (#number-formatting)

Supported chart types

Data preprocessed by Flexmonster Connector for amCharts is of the array of objects (https://www.amcharts.com/docs/v4/concepts/data/#Structure_of_data) format. All amCharts chart types accept this data format, so Flexmonster supports all of them.

See how to integrate the component with different chart types:

- Area chart (demo (https://jsfiddle.net/flexmonster/gjv5m3y8/))
- Bar chart (demo (https://jsfiddle.net/flexmonster/91jto86a/))
- Bubble chart (demo (https://jsfiddle.net/flexmonster/pnbtofxk/))
- Clustered bar chart (demo (https://jsfiddle.net/flexmonster/tjdcL1rs/))
- Clustered column chart (demo (https://jsfiddle.net/flexmonster/sfdvhu2m/))
- Column chart (demo (https://jsfiddle.net/flexmonster/k5v9c1me/))
- Donut chart (demo (https://jsfiddle.net/flexmonster/2smxeL7t/))
- Line chart (demo (https://jsfiddle.net/flexmonster/w5nosrug/))
- Nested donut (demo (https://jsfiddle.net/flexmonster/kt2wy5un/))
- Pie chart (demo (https://jsfiddle.net/flexmonster/8uvbqoz4/))
- Radar chart (demo (https://jsfiddle.net/flexmonster/fu7g43te/))
- Radar chart with switched axes (demo (https://jsfiddle.net/flexmonster/L3mkczq6/))
- Semi-circle pie chart (demo (https://jsfiddle.net/flexmonster/3z7d4npx/))
- Stacked bar chart (demo (https://jsfiddle.net/flexmonster/u4jxs5he/))

- 100% Stacked column chart (demo (https://jsfiddle.net/flexmonster/ujp9n4sk/))
- 3d Donut chart (demo (https://jsfiddle.net/flexmonster/ud3fx169/))

More ready-to-use examples of integration with amCharts you can find on the Examples page (https://www.flexmonster.com/examples/#!amcharts).

Adding amCharts

The steps below describe how to create a pie chart (https://www.amcharts.com/docs/v4/chart-types/pie-chart/) based on data received from the component. To integrate Flexmonster with other chart types, refer to the amCharts documentation (https://www.amcharts.com/docs/v4/).

Step 1. Embed the component into your web page

Add Flexmonster to your web page and configure a simple report (e.g., based on a JSON or CSV data source). Replace "XXXX-XXXX-XXXX-XXXX-XXXX" with your license key. If you don’t have a license key, contact our team (https://www.flexmonster.com/contact/) and request a special trial key.

```
<div id="pivotContainer">The component will appear here</div>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>

<script>
    var pivot = new Flexmonster({
        container: "pivotContainer",
        componentFolder: "https://cdn.flexmonster.com/",
        height: 300,
        report: {
            dataSource: {
                filename: "data/data.csv"
            },
            slice: {
                rows: [
                    { uniqueName: "Country" }
                ],
                columns: [
                    { uniqueName: "Color" },
                    { uniqueName: "[Measures]" }
                ],
                measures: [
                    { uniqueName: "Price" }
                ]
            }
        },
        licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
    });
</script>
```

Step 2. Add amCharts files

To use the amCharts functionality, include amCharts files to your web page:

```
<script src="https://www.amcharts.com/lib/4/core.js"></script>
```

```
<script src="https://www.amcharts.com/lib/4/charts.js"></script>
<script src="https://www.amcharts.com/lib/4/themes/animated.js"></script>
```

Including files directly to the web page is one of the possible ways to add amCharts. Refer to the amCharts installation guide (https://www.amcharts.com/docs/v4/getting-started/basics/#Installation) for other options of adding amCharts.

Step 3. Add Flexmonster Connector for amCharts

Flexmonster Connector for amCharts provides ready-to-use methods for easy and smooth integration with amCharts. Include it to your web page with the following lines of code:

```
<script
    src="https://cdn.flexmonster.com/lib/flexmonster.amcharts.js">
</script>
```

Step 4. Add a container for amCharts

Add a container for the chart:

```
<div id="amchartsContainer"></div>
```

Step 5. Add a reportcomplete event handler

A chart should be created only when the component is ready to provide the data. To track this, use the reportcomplete event:

```
var pivot = new Flexmonster({
    container: "#pivotContainer",
    componentFolder: "https://cdn.flexmonster.com/",
    report: {
       // the report from the 1st step
    },
    licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX",
    reportcomplete: function() {
        pivot.off("reportcomplete");
        createChart();
    }
});
```

When data loading is complete, the event handler will invoke the createChart() function to draw the chart.

The implementation of the createChart() function is given below.

Step 6. Create a variable for the chart

Declare a variable to work with the chart instance:

```
let chart;
```

The variable should be visible to functions that draw and redraw the chart.

Step 7. Request data from the component

The Connector for amCharts extends the Flexmonster API with the amcharts.getData() method, which requests data from the component and structures it to the format required by amCharts (https://www.amcharts.com/docs/v4/concepts/data/#Structure_of_data). Call the amcharts.getData() method to get data from the component:

```
function createChart() {
    pivot.amcharts.getData({}, drawChart, updateChart);
}
```

Data returned by the amcharts.getData() method contains fields specified in the report.slice object. If the amcharts.getData() method gets the slice as a parameter, the data is prepared according to it.

See more details about the amcharts.getData() (https://www.flexmonster.com/api/amcharts-getdata/) method.

Step 8. Create a function to draw the chart

The drawChart() function initializes the chart, sets configurations needed for it, and fills the chart with data provided by the amcharts.getData() method:

```
function drawChart(chartData, rawData) {
    // initialize the chart
    var chart = am4core.create("amchartsContainer", am4charts.PieChart);
    //fill the chart with the data from Flexmonster
    chart.data = chartData.data;
    chart.legend = new am4charts.Legend();
    // Create pie series
    var series = chart.series.push(new am4charts.PieSeries());
    series.dataFields.category = pivot.amcharts.getCategoryName(rawData);
    series.dataFields.value = pivot.amcharts.getMeasureNameByIndex(rawData, 0);
}
```

Notice the following lines in the code snippet:

```
series.dataFields.category = pivot.amcharts.getCategoryName(rawData);
series.dataFields.value = pivot.amcharts.getMeasureNameByIndex(rawData, 0);
```

The amcharts.getCategoryName() method is used to set the category name for the amCharts Category axis (https://www.amcharts.com/docs/v4/concepts/axes/category-axis/). Learn more about how the method chooses the category name (#choose-category).

Then, getMeasureNameByIndex() sets the value for the Value axis (https://www.amcharts.com/docs/v4/concepts/axes/value-axis/).

For more details on how the pie chart is created, see the amCharts documentations (https://www.amcharts.com/docs/v4/chart-types/pie-chart/).

Step 9. Create a function to update the chart

Create a function that will redraw the chart once the report is updated (e.g., when the data is filtered, sorted, and so on):

```
function updateChart(chartData, rawData) {
    chart.dispose();
    drawChart(chartData, rawData);
}
```

When the data is updated, the function will dispose of the current chart and draw a new chart based on the updated data.

Step 10. See the results

Launch your web page from the browser and see an interactive pie chart displaying data from the component. Check out a live example on JSFiddle (https://jsfiddle.net/flexmonster/8uvbqoz4/).

Preparing data for the chart

This section explains how Flexmonster Connector for amCharts preprocesses data for the chart.

The amcharts.getData() method returns data in the format required by amCharts (https://www.amcharts.com/docs/v4/concepts/data/#Structure_of_data), namely an array of objects. For example:

```
[
    {
        "categoryName": "value",
        "measureName 1": "value",
         …
        "measureName n": "value",
    }
    …
]
```

The object contains only one category field and all the measures presented in the slice (https://www.flexmonster.com/doc/slice/).

The field to represent the category is chosen as follows:

1. If the slice.rows (https://www.flexmonster.com/api/slice-object/#rows) array contains some fields, the first field from rows is chosen to be the category.
2. If the slice.rows array is empty and the slice.columns (https://www.flexmonster.com/api/slice-object/#columns) array contains some fields, the first field from columns is chosen as the category.

If both slice.rows and slice.columns are empty, then no category is available, and amcharts.getCategoryName() will return undefined.

In such cases, the getMeasureNameByIndex() method can be used to select the category for series.

Configuring number formatting

The Connector allows using a number format set via the Format Object (https://www.flexmonster.com/api/format-object/) for amCharts as well. The amcharts.getNumberFormatPattern() method converts the Format Object (https://www.flexmonster.com/api/format-object/) received from the component to the amCharts number formatting string.

When the formatting string is prepared by the method, the following properties of the Format Object are considered:

- decimalPlaces
- maxDecimalPlaces
- negativeNumberFormat (only the -1 and (1) formats are available)
- currencySymbol
- positiveCurrencyFormat
- negativeCurrencyFormat (only the $-1, -1$, ($1), (1$) formats are available)
- isPercent

Regardless of configurations set in the Format Object, the thousandsSeparator and decimalSeparator format parameters always have constant values. These values are , for thousandsSeparator and . for decimalSeparator. amCharts imposes this limitation as it uses predefined separators for decimals and thousands (https://www.amcharts.com/docs/v4/concepts/formatters/formatting-numbers/#Format_codes).

Here is an example of the formatting string returned by the amcharts.getNumberFormatPattern() method:

'$'#,###.00|'('$'#,###.00')'

The part of the string before | defines the positive number format. The part of the string after | defines the negative number format. For example, 1205 will be shown as $1,205.00, and -1205 will be shown as ($1,205.00).

What’s next?

You may be interested in the following articles:

- API reference of Flexmonster Connector for amCharts (/api/all-methods-amcharts/)
- How to configure the data source (https://www.flexmonster.com/doc/data-source/)
- How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/)
- How to add localization (https://www.flexmonster.com/doc/localizing-component/)
- How to customize the appearance with CSS (https://www.flexmonster.com/doc/customizing-appearance/)

7.5. Integration with Google Charts

This tutorial will help you integrate Flexmonster with the Google Charts (https://developers.google.com/chart/)

charting library.

This guide contains the following sections:

- Supported chart types (#supported-charts)
- Adding Google Charts (#adding-google-charts)
- Flexmonster Connector for Google Charts (#google-charts-connector)

Supported chart types

Flexmonster supports the following Google chart types:

- AreaChart (demo (https://jsfiddle.net/flexmonster/gwgueexf/))
- BarChart (demo (https://jsfiddle.net/flexmonster/gd5pLvmm/))
- ColumnChart (demo (https://jsfiddle.net/flexmonster/xdpqw7k7/))
- GeoChart (demo (https://jsfiddle.net/flexmonster/u8vsd3z7/))
- LineChart (demo (https://jsfiddle.net/flexmonster/8378ax2z/))
- PieChart (demo (https://jsfiddle.net/flexmonster/ygf8ekk6/))
- Sankey (demo (https://jsfiddle.net/flexmonster/jpy8vayk/))

If the chart type that you want to use is not on the list, it is possible to use prepareDataFunction to preprocess the data to fit your preferences.

More ready-to-use examples of integration with Google Charts you can find on the Examples page (https://www.flexmonster.com/examples/#!google-charts).

Adding Google Charts

1. Add the component using data from a CSV file to your HTML page. Replace XXXX-XXXX-XXXX-XXXX-XXXX with your license key. If you don't have a license key, contact our team (https://www.flexmonster.com/contact/) and request a special trial key.

```
<div id="pivotContainer">The component will appear here</div>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>

<script>
    var pivot = new Flexmonster({
        container: "pivotContainer",
        componentFolder: "https://cdn.flexmonster.com/",
        toolbar: true,
        report: {
            dataSource: {
                filename: "data.csv"
            },
            slice: {
                rows: [
                    { uniqueName: "Country" }
                ],
                columns: [
                    { uniqueName: "[Measures]" }
                ],
                measures: [
                    { uniqueName: "Price" }
                ]
            }
        },
        licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
    });
</script>
```

2. Add Google Charts:

   ```
   <script src="https://www.gstatic.com/charts/loader.js"></script>
   ```

3. Add Flexmonster Connector for Google Charts:

   ```
   <script src="https://cdn.flexmonster.com/lib/flexmonster.googlecharts.js"></sc
   ript>
   ```

4. Add a container for the chart:

   ```
   <div id="googlechartContainer"></div>
   ```

5. Add a reportComplete event handler to know when the pivot table is ready to be a data provider:

   ```
   reportcomplete: function() {
       pivot.off("reportcomplete");
       pivotTableReportComplete = true;
       createGoogleChart();
   }
   ```

6. Add a function to create the chart. This function uses the Connector for Google Charts from flexmonster.googlecharts.js. The Connector extends the Flexmonster API with the following call: googlecharts.getData(options, callbackHandler, updateHandler).

   ```
   var pivotTableReportComplete = false;
   var googleChartsLoaded = false;

   google.charts.load('current', {'packages':['corechart']});
   google.charts.setOnLoadCallback(onGoogleChartsLoaded);

   function onGoogleChartsLoaded() {
       googleChartsLoaded = true;
       if (pivotTableReportComplete) {
           createGoogleChart();
       }
   }

   function createGoogleChart() {
       if (googleChartsLoaded) {
           pivot.googlecharts.getData({ type: "column" },
               drawChart,
               drawChart
           );
       }
   }

   function drawChart(chartConfig) {
       var data = google.visualization.arrayToDataTable(chartConfig.data);
       var options = {
   ```

```
            title: chartConfig.options.title,
            height: 300
        };
        var chart = new google.visualization
            .ColumnChart(document.getElementById('googlechartContainer'));
        chart.draw(data, options);
    }
```

You will see a column chart that displays the same data that is shown in the pivot pivot table instance and it will react to updates. Check out a live example on JSFiddle (https://jsfiddle.net/flexmonster/xdpqw7k7/).

Flexmonster Connector for Google Charts

flexmonster.googlecharts.js is a connector between our component and Google Charts.
The googlecharts.getData() method requests data from the pivot table and preprocesses it to the appropriate format for the required type of chart.

googlecharts.getData(options:Object, callbackHandler:Function, updateHandler:Function) has the following parameters:

- options – Object. This object has the following properties:
    - type (optional) – String. The chart type to prepare data for. Possible values: "area", "bar", "column", "line", "pie", "sankey". If the type is not specified, the data will be prepared the same way as for "sankey".
    - slice (optional) – Object. Defines the slice from which the data should be returned (for JSON and CSV data sources only). If not defined, the API call will return data currently displayed in the pivot table.

      ```
      function createGoogleChart() {
          if (googleChartsLoaded) {
              pivot.googlecharts.getData(
                  {
                      slice: {
                          rows: [{uniqueName: "Country"}],
                          columns: [{uniqueName: "[Measures]"}],
                          measures: [{uniqueName: "Quantity"}]
                      }
                  },
                  drawChart,
                  drawChart
              );
          }
      }
      ```

    - prepareDataFunction (optional) – An external function. If the Connector does not include the necessary type of chart or if you need to preprocess the data differently, use this function. prepareDataFunction takes two input parameters: rawData – the raw data (check out the structure of rawData in getData() (52.207.238.113/api/getdata/)); options – an object with options set in googlecharts.getData(). Check out the example on JSFiddle (https://jsfiddle.net/flexmonster/rzrbpyc2/) based on the data preprocessed with this external function. One more example (https://jsfiddle.net/flexmonster/aqe5qne8/) illustrates how the data can be prepared to show the deepest drill-down level on Google Column Chart.
- callbackHandler – Function. Specifies what will happen once the data is ready. Additional options can be specified and then data can be passed directly to the charting library. Takes two input parameters –

chartConfig and rawData (rawData is passed just in case you need it, for example for defining number formatting in the tooltip).
- updateHandler (optional) – Function. Takes two input parameters – chartConfig and rawData. Specifies what will happen once data in the pivot table is filtered/sorted/etc or a number format is changed.

The Connector has several functions for defining number formatting for Google Charts. All these functions take the pivot table format object and return the formatting string in Google Charts format.

- googlecharts.getNumberFormat(format) – Object. Takes the pivot table format object and returns a format object for number formatting in Google Charts. This object can be used to format columns in DataTable.
- googlecharts.getNumberFormatPattern(format) – Object. Takes the pivot table format object and returns a Google Charts format pattern. This pattern can be used to format axes.

These functions can be used in callbackHandler and updateHandler to define a number formatting for axes and tooltips. Try it on JSFiddle (https://jsfiddle.net/flexmonster/dkerf354/).

7.6. Integration with FusionCharts

This tutorial will help you integrate Flexmonster with the FusionCharts (https://www.fusioncharts.com/) charting library.

This guide contains the following sections:

- Supported chart types (#supported-charts)
- Adding FusionCharts (#adding-fusioncharts)
- Flexmonster Connector for FusionCharts (#fusioncharts-connector)

Supported chart types

Flexmonster supports the following FusionCharts types:

- area2d (demo (https://jsfiddle.net/flexmonster/09j1jmba/))
- bar2d (demo (https://jsfiddle.net/flexmonster/hwbxzmkd/))
- bar3d (demo (https://jsfiddle.net/flexmonster/wdLzeyfp/))
- column2d (demo (https://jsfiddle.net/flexmonster/2gshr9aL/))
- column3d (demo (https://jsfiddle.net/flexmonster/w2hhdL6a/))
- doughnut2d (demo (https://jsfiddle.net/flexmonster/2qdkaj4g/))
- doughnut3d (demo (https://jsfiddle.net/flexmonster/fzg4zuac/))
- line (demo (https://jsfiddle.net/flexmonster/3tu6pmh9/))
- marimekko (demo (https://jsfiddle.net/flexmonster/1xhod06u/))
- msarea (demo (https://jsfiddle.net/flexmonster/bj40dt9e/))
- msbar2d (demo (https://jsfiddle.net/flexmonster/vmacrbr7/))
- msbar3d (demo (https://jsfiddle.net/flexmonster/L1gfre6b/))
- mscolumn2d (demo (https://jsfiddle.net/flexmonster/heqp5me8/))
- mscolumn3d (demo (https://jsfiddle.net/flexmonster/ktkgxz3q/))
- mscolumn3dlinedy (demo (https://jsfiddle.net/flexmonster/p4fyh8sr/))
- msline (demo (https://jsfiddle.net/flexmonster/3tqm4gq9/))
- msspline (demo (https://jsfiddle.net/flexmonster/zdjgtx0j/))
- mssplinearea (demo (https://jsfiddle.net/flexmonster/whL2rLy1/))
- pareto2d (demo (https://jsfiddle.net/flexmonster/vou8zueo/))
- pareto3d (demo (https://jsfiddle.net/flexmonster/ej4vffo9/))
- pie2d (demo (https://jsfiddle.net/flexmonster/66hra3jL/))
- pie3d (demo (https://jsfiddle.net/flexmonster/yrm6bpw9/))
- radar (demo (https://jsfiddle.net/flexmonster/8g0ehgxs/))

- spline (demo (https://jsfiddle.net/flexmonster/5p6uL3wL/))
- splinearea (demo (https://jsfiddle.net/flexmonster/zuwc00oc/))
- stackedarea2d (demo (https://jsfiddle.net/flexmonster/sqvLps37/))
- stackedbar2d (demo (https://jsfiddle.net/flexmonster/6L6d4cLy/))
- stackedcolumn2d (demo (https://jsfiddle.net/flexmonster/qh5wpm71/))
- stackedcolumn3d (demo (https://jsfiddle.net/flexmonster/qjufLt2u/))
- Supported map types:maps/worldwithcountries (demo (https://jsfiddle.net/flexmonster/Ludwah6k/))

If the chart type that you want to use is not on the list, it is possible to use prepareDataFunction to preprocess the data to fit your preferences.

More ready-to-use examples of integration with FusionCharts you can find on the Examples page (https://www.flexmonster.com/examples/#!fusioncharts).

Adding FusionCharts

1. Add the component using data from a CSV file to your HTML page. Replace XXXX-XXXX-XXXX-XXXX-XXXX with your license key. If you don’t have a license key, contact our team (https://www.flexmonster.com/contact/) and request a special trial key.

```
<div id="pivotContainer">The component will appear here</div>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>

<script>
    var pivot = new Flexmonster({
        container: "pivotContainer",
        componentFolder: "https://cdn.flexmonster.com/",
        toolbar: true,
        report: {
            dataSource: {
                filename: "data.csv"
            },
            slice: {
                rows: [
                    { uniqueName: "Country" }
                ],
                columns: [
                    { uniqueName: "Business Type" },
                    { uniqueName: "[Measures]" }
                ],
                measures: [
                    { uniqueName: "Price" }
                ]
            }
        },
        licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
    });
</script>
```

2. Add FusionCharts:

```
<script src="https://static.fusioncharts.com/code/latest/fusioncharts.js"></sc
ript>
```

3. Add Flexmonster Connector for FusionCharts:

```
<script src="https://cdn.flexmonster.com/lib/flexmonster.fusioncharts.js"></sc
ript>
```

4. Add a container for the chart:

   ```
   <div id="fusionchartContainer"></div>
   ```

5. Add a reportComplete event handler to know when the pivot table is ready to be a data provider:

   ```
   reportcomplete: function() {
       pivot.off("reportcomplete");
       createFusionChart();
   }
   ```

6. Add a function to create the chart. This function uses the Connector for FusionCharts from flexmonster.fusioncharts.js. The Connector extends the Flexmonster API with the following call: fusioncharts.getData(options, callbackHandler, updateHandler).

   ```
   function createFusionChart() {
       var chart = new FusionCharts({
           "type": "doughnut2d",
           "renderAt": "fusionchartContainer"
       });

       pivot.fusioncharts.getData({
           type: chart.chartType()
       }, function(chartConfig) {
           chart.setJSONData(chartConfig);
           chart.render();
       }, function(chartConfig) {
           chart.setJSONData(chartConfig);
       });
   }
   ```

You will see a doughnut2d chart that displays the same data that is shown in the pivot pivot table instance and it will react to updates. Try it on JSFiddle (https://jsfiddle.net/flexmonster/2qdkaj4g/).

Flexmonster Connector for FusionCharts

flexmonster.fusioncharts.js is a connector between our component and FusionCharts. The fusioncharts.getData() method requests data from the pivot table and preprocesses it to the appropriate format for the required type of chart.

fusioncharts.getData(options:Object, callbackHandler:Function, updateHandler:Function) has the following parameters:

- options – Object. This object has the following properties:
  - type – String. The chart type to prepare data for. This property is required.
  - slice (optional) – Object. Defines the slice from which the data should be returned (for JSON and CSV data sources only). If not defined, the API call will return data currently displayed in the pivot table.

    ```
    function createFusionChart() {
        var chart = new FusionCharts({
    ```

```
                "type": "doughnut2d",
                "renderAt": "fusionchartContainer"
            });

            pivot.fusioncharts.getData({
                type: chart.chartType(),
                slice: {
                    rows: [{uniqueName: "Country"}],
                    columns: [{uniqueName: "[Measures]"}],
                    measures: [{uniqueName: "Price"}, {uniqueName: "Discount"}]

                },
            }, function(chartConfig) {
                chart.setJSONData(chartConfig);
                chart.render();
            }, function(chartConfig) {
                chart.setJSONData(chartConfig);
            });
        }
```

    - prepareDataFunction (optional) – An external function. If the Connector does not include the necessary type of chart or if you need to preprocess the data differently, use this function. prepareDataFunction takes two input parameters: rawData – the raw data (check out the structure of rawData in getData() (52.207.238.113/api/getdata/)); options – an object with options set in fusioncharts.getData(). Try it on JSFiddle (https://jsfiddle.net/flexmonster/sytk76tu/).
- callbackHandler – Function. Specifies what will happen once data is ready. Additional options can be specified and then data can be passed directly to the charting library. Takes two input parameters – chartConfig and rawData. chartConfig is ready to be used in the chart. rawData is passed just in case you need access to rawData.meta properties (check out the structure of rawData in getData() (/api/getdata/)), for example to define number formatting.
- updateHandler (optional) – Function. Takes two input parameters – chartConfig and rawData. Specifies what will happen once data in the pivot table is filtered/sorted/etc or a number format is changed.

The Connector has an API call for defining the number formatting for FusionCharts: fusioncharts.getNumberFormat(format:Object) – Object. Takes a pivot table format object and returns a format object for number formatting in FusionCharts. You may need this call when you are defining your own prepareDataFunction and want to use number formatting from the pivot table on the chart. The returned object has the following parameters:

- decimalSeparator
- decimals
- forceDecimals
- numberPrefix
- thousandSeparator

They can be used as is in chart object for FusionCharts. Here is an example of using the fusioncharts.getNumberFormat call inside prepareDataFunction:

```
var format =
    pivot.fusioncharts.getNumberFormat(data.meta.formats[0]);
for (var prop in format) {
    output.chart[prop] = format[prop];
}
```

If you need to define a number format for the second Y axis for FusionChart, you can just add a "s" prefix to each property of the returned format object when copying them to the chart object, as follows:

```
var format2 =
    pivot.fusioncharts.getNumberFormat(data.meta.formats[1]);
for (var prop in format2) {
    output.chart["s"+prop] = format2[prop];
}
```

Try it on JSFiddle (https://jsfiddle.net/flexmonster/mky56ghy/).

7.7. Integration with any charting library

This tutorial will help you connect a 3rd party visualization tool to Flexmonster Pivot Table and Charts. This simple example is based on d3.js (http://bl.ocks.org/mbostock/3885304) and aims to illustrate the interaction between data from Flexmonster and external visualization. Integration with any other library will have similar basic steps.

The integration is based on the getData() (/api/getdata/) API call. Read about it to understand the format that the data is returned in from the component. In this article we will connect the pivot table data with the d3.js chart step by step:

Adding the basis for a new chart

1. Add the component using data from a CSV file to your HTML page. Replace XXXX-XXXX-XXXX-XXXX-XXXX with your license key. If you don’t have a license key, contact our team (https://www.flexmonster.com/contact/) and request a special trial key.

```
<div id="pivotContainer">The component will appear here</div>
<script src="https://cdn.flexmonster.com/flexmonster.js"></script>

<script>
    var pivot = new Flexmonster({
        container: "pivotContainer",
        componentFolder: "https://cdn.flexmonster.com/",
        toolbar: true,
        report: {
            dataSource: {
                filename: "data.csv"
            },
            slice: {
                rows: [
                    { uniqueName: "Country" }
                ],
                columns: [
                    { uniqueName: "Business Type" },
                    { uniqueName: "[Measures]" }
                ],
                measures: [
                    { uniqueName: "Price" }
                ]
            }
        },
        licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
    });
</script>
```

2. Add a container for the chart:

   ```
   <svg id="d3Chart" width="650" height="230"></svg>
   ```

3. Add a reportCompleteevent handler to know when the pivot table is ready to be a data provider:

   ```
   reportcomplete: function() {
       pivot.off("reportcomplete");
       createChart();
   }
   ```

4. Add a function to create the chart. This function uses getData(options, callbackHandler, updateHandler).

   ```
   function createChart() {
       pivot.getData(
           {
               // define your slice
           },
           drawChart,
           updateChart
       );
   }
   ```

Try it on JSFiddle (https://jsfiddle.net/flexmonster/ybaefLh4/).

Preparing the data and drawing the chart

The most important part of drawing a chart is preparing the data by transforming it from the format returned by the getData() API call to the format that suits the 3rd party visualization tool:

```
var data = prepareDataFunction(rawData);
```

This example shows how to define and use a function (in our example it is prepareDataFunction) to process the data. This function should prepare data appropriately for the charting library format. In this example (https://jsfiddle.net/flexmonster/ybaefLh4/) prepareDataFunction iterates through the data array from rawData and discards a record containing the grand total because it is unnecessary for the bar chart. The function also renames rows from r0 to member and values from v0 to value. This is not required, but it makes the code more readable when referring to the data later. We have the following pivot table:

Country	Total Sum of Price
Australia	1 372 281
France	1 117 794
Germany	1 070 453
Canada	1 034 112
United States	847 331
United Kingdom	779 899
Grand Total	6 221 870

The data array from rawData looks like this:

```
data:[
 {
  v0:6221870
 },
 {
  r0:"Australia",
  v0:1372281
 },
 {
  r0:"France",
  v0:1117794
 },
 {
  r0:"Germany",
  v0:1070453
 },
 {
  r0:"Canada",
  v0:1034112
 },
 {
  r0:"United States",
  v0:847331
 },
 {
  r0:"United Kingdom",
  v0:779899
 }
]
```

After prepareDataFunction the data will look like this:

```
        {
            member:"Australia",
            value:1372281
        },
        {
            member:"France",
            value:1117794
        },
        {
            member:"Germany",
            value:1070453
        },
        {
            member:"Canada",
            value:1034112
        },
        {
            member:"United States",
            value:847331
        },
        {
            member:"United Kingdom",
            value:779899
        }
```

The drawChart function draws a chart using the processed data. In our JSFiddle example (https://jsfiddle.net/flexmonster/ybaefLh4/), the logic of drawing is the same as in the d3.js example (http://bl.ocks.org/mbostock/3885304). The updateChart function works similarly but clears the SVG first.

8. Customizing

8.1. Available tutorials

In Flexmonster, you can modify the Toolbar and context menus, customize the appearance, and change localization.

To assist you in the customization process, we created a set of step-by-step guides. Let’s have a brief look at the tutorials you can find in this section.

Customizing the Toolbar

The Toolbar provides access to the most common features. It is highly customizable: you can configure the existing tabs or add custom ones. See our tutorial to learn more: Customizing the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/).

Customizing appearance

To change the component’s look and feel, you can use one of the predefined color schemes or create a custom theme. Refer to our guide for more details: Customizing appearance (https://www.flexmonster.com/doc/customizing-appearance/).

Customizing the context menu

Every cell or chart element in Flexmonster has a context menu. Our tutorial describes how to change it: Customizing the context menu (https://www.flexmonster.com/doc/customizing-context-menu/).

Customizing the grid

This guide covers the cases when grid cells need to be highlighted based on the data, position, or type: Customizing the grid (https://www.flexmonster.com/doc/customizing-grid/).

Customizing the pivot charts

Visit this guide if you need to set custom colors or captions for the pivot charts: Customizing the pivot charts (https://www.flexmonster.com/doc/customizing-pivot-charts/).

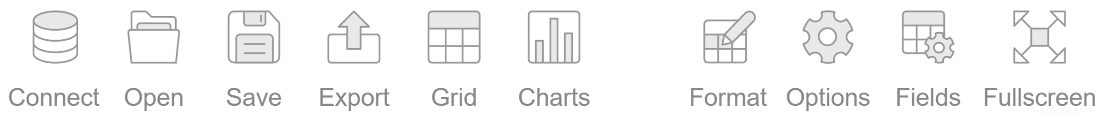
Localizing the component

You can translate Flexmonster Pivot to any language: Localizing the component (https://www.flexmonster.com/doc/localizing-component/).

8.2. Customizing the Toolbar

About the Toolbar

The Toolbar is an HTML/JS addition to Flexmonster. It uses a standard API (https://www.flexmonster.com/api/) and provides easy access to the most commonly used features. The Toolbar is free and provided “as is”.



The Toolbar is available since version 2.0. Enabling the Toolbar is very easy, just set the toolbar parameter in the new Flexmonster() function call to true:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true
});
```

Ensure that your flexmonster/ folder includes the toolbar/ folder and that it’s not empty.

Use flexmonster.toolbar to get a reference to the Toolbar instance. It allows you to call its functions on the page from outside of Flexmonster Pivot.

Customizing

The Toolbar can be customized using the beforetoolbarcreated (https://www.flexmonster.com/api/beforetoolbarcreated/) event. Tabs and buttons can be removed from it and new ones can be easily added.

Below we describe how to perform basic Toolbar customization. See the Examples page (https://www.flexmonster.com/examples/#!customizing-toolbar) for more examples on how to modify the Toolbar.

Removing a tab from the Toolbar

Add a beforetoolbarcreated event handler. Inside the handler you can get all tabs using the getTabs() method that returns an Array of Objects, each of which describes a tab. To remove a tab just delete the corresponding Object from the Array. The following example will remove the first tab:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    beforetoolbarcreated: customizeToolbar
});
```

```
function customizeToolbar(toolbar) {
    // get all tabs
    var tabs = toolbar.getTabs();
    toolbar.getTabs = function () {
        // delete the first tab
        delete tabs[0];
        return tabs;
    }
}
```

Open the example on JSFiddle (https://jsfiddle.net/flexmonster/x2tpc31w/).

Adding a new tab to the Toolbar

The following code will add a new tab:

```
var pivot = new Flexmonster({
    container: "pivotContainer",
    toolbar: true,
    beforetoolbarcreated: customizeToolbar
});

function customizeToolbar(toolbar) {
    // get all tabs
    var tabs = toolbar.getTabs();
    toolbar.getTabs = function () {
        // add new tab
        tabs.unshift({
            id: "fm-tab-newtab",
            title: "New Tab",
            handler: newtabHandler,
            icon: this.icons.open
        });
        return tabs;
    }
    var newtabHandler = function() {
         // add new functionality
    }
}
```

where:

- title – String. The tab’s label.
- id – String. The id used in CSS styles.
- handler – Function. The function that handles clicks on this tab.
- icon – String. The HTML tag containing your custom icon for this new tab. You can choose one of the basic vector icons defined in the flexmonster.toolbar.js file.

There are also some optional parameters:

- args – Any. Arguments to pass to the handler.

- menu – Array. Dropdown menu items.
- mobile – Boolean. When set to false, the tab does not show on mobile devices. *Default value: true*.
- ios – Boolean. When set to false, the tab does not show on iOS devices. *Default value: true*.
- android – Boolean. When set to false false, the tab does not show on Android devices. *Default value: true*.
- rightGroup – Boolean. When set to true, the tab is positioned on the right side of the Toolbar. *Default value: false*.

Check out an example of creating a new tab on JSFiddle (https://jsfiddle.net/flexmonster/m7e74ay4/).

Further customization

You can customize almost everything. To explore all the options, we recommend investigating the existing code. Look in the toolbar/ folder (you can find it in **[package]/**flexmonster/). Open the flexmonster.toolbar.js file. Find the tab section (it starts with the getTabs() function expression) to understand how it works.

To change the appearance of the Toolbar, read customizing appearance (https://www.flexmonster.com/doc/customizing-appearance/).

What's next?

You may be interested in the following articles:

- How to add localization (/doc/localizing-component/)
- How to define a format for date and time (/doc/date-and-time-formatting/)
- How to set specific options common for all reports (/doc/global-object/)
- How to configure the way that data is exported (/doc/export-and-print/)

8.3. Customizing appearance

You can customize the appearance of Flexmonster using CSS — the same way as for regular HTML. Flexmonster offers predefined skins and provides their source code so you can quickly create custom ones.

Built-in themes

Flexmonster comes with a variety of predefined CSS themes:

- Striped-Blue (https://jsfiddle.net/flexmonster/cnxgbrqj/)
- Striped-Teal (https://jsfiddle.net/flexmonster/Lvws2n7h/)
- Purple (https://jsfiddle.net/flexmonster/40vdyuoz/)
- Black-Orange (https://jsfiddle.net/flexmonster/49ot3mpg/)
- Bright-Orange (https://jsfiddle.net/flexmonster/zgmynkx8/)
- Yellow (https://jsfiddle.net/flexmonster/zpf3e0a6/)
- Green (https://jsfiddle.net/flexmonster/yvj72uen/)
- Midnight (https://jsfiddle.net/flexmonster/tf1pg9a2/)
- Mac OS (https://jsfiddle.net/flexmonster/6dh2cjLw/)
- Soft-Default (https://jsfiddle.net/flexmonster/vdne3kf6/)
- Accessible (https://jsfiddle.net/flexmonster/zebrn9sL/) (compatible with high contrast modes)
- Light blue (https://jsfiddle.net/flexmonster/j8nxrusp/)
- Dark (https://jsfiddle.net/flexmonster/73bzurqd/)
- Teal (https://jsfiddle.net/flexmonster/xvtbq87d/)
- Orange (https://jsfiddle.net/flexmonster/t8Ls0dgp/)
- Default (https://jsfiddle.net/flexmonster/27aocdep/)
- 2.3-styled (https://jsfiddle.net/flexmonster/e2krzvq2/) (for those who enjoyed the style of major version 2.3)

Explore all available themes inside the flexmonster/theme/ folder or check out the Flexmonster themes demo (/demos/themes/). If no theme is specified, the component uses the default theme. Its CSS is available inside the flexmonster/flexmonster.css and flexmonster/flexmonster.min.css files. To apply a different theme, add the reference to the minified CSS file of the chosen theme. For example, to apply the lightblue theme, insert the following line of code:

```
<link rel="stylesheet" type="tex
t/css" href="/theme/lightblue/flexmonster.min.css" />
```

To insert some other theme, just replace lightblue from the CSS reference with the name of the chosen theme. The 2.3-styled theme can be set like this:

```
<link rel="stylesheet" type="text/css" href="/theme/old/flexmonster.min.css" />
```

Check out the Flexmonster default theme in the pivot table demo (/demos/pivot-table-js/).

Adding a custom theme

You can create your custom theme using the custom theme builder or manually.

Create using the theme builder

This guide will help you create a custom theme for Flexmonster using the theme builder.

Step 1. Download the .zip archive with the theme builder or clone it from GitHub (https://github.com/flexmonster/custom-theme-builder) with the following commands:

```
git clone https://github.com/flexmonster/custom-theme-builder
cd custom-theme-builder
```

Step 2. Install the npm packages described in package.json:

```
npm install
```

Step 3. In the custom-theme-builder/ folder, find and open the flexmonster.less file. It contains styles for Flexmonster. Feel free to adjust variable values and colors to your needs.

For example, let’s change font styles and the theme main color:

```
@font-family: serif;
@font-size: 15px;
...
@theme-color: antiquewhite;
```

Other CSS styles can be customized similarly.

Step 4. Run the custom theme builder to generate CSS and minified CSS files for the theme:

```
npm start
```

As a result, the generated-theme/ folder with your custom theme files (flexmonster.css, flexmonster.min.css) will appear in custom-theme-builder/.

Step 5. Apply your theme to the component by including the CSS or minified CSS file:

```
<link rel="stylesheet" type="text/css"
 href="your-path\custom-theme-builder\generated-theme\flexmonster.min.css"
/>
```

Now launch the page from a browser — your custom theme is set for the component.

Create manually

Creating a custom theme manually involves the following steps:

Step 1. Inside the theme/ folder, make a copy of any folder with an existing theme (e.g., lightblue/).

Step 2. Rename the copied folder to a custom theme name.

Step 3. Replace the theme's colors with your custom ones. There are several ways to achieve this:

- The recommended way requires using a CSS pre-processor named Less (https://lesscss.org/). Set custom colors inside the flexmonster.less file from the custom theme's folder.
  After replacing the necessary colors, compile flexmonster.less into flexmonster.css and flexmonster.min.css.
  More details about this compilation are available in the Less documentation (https://lesscss.org/#using-less).
- The other option is to edit the colors inside the flexmonster.css file from your theme folder. This approach is not recommended because it complicates updating your own theme with the updates made in the component CSS.

Step 4. Add the reference to the CSS or to the minified CSS. This will apply your new theme:

```
<link rel="stylesheet" type="text/css"
 href="/theme/your-new-theme/flexmonster.css"
/>
```

Further customization

To add custom CSS above basic styling, we suggest one of the following approaches:

- If you use Less, we advise creating your own theme based on one of the existing ones. Create your own theme with the help of the adding your own theme (#new-theme) guide. Then you can open your theme/your-new-theme/flexmonster.less file and write custom code at the bottom of this file, after the definition of the variables. This way our base Less file theme/flexmonster-base.less will remain unchanged and will not cause any issues during updating. Check out the example with custom Less code added on top of the orange theme. Open the theme/orange/flexmonster.less file and find the following lines:

  ```
  #fm-pivot-view .fm-grid-layout div.fm-header {
      border-right: 1px solid @theme-color-supersuperlight;
      border-bottom: 1px solid @theme-color-supersuperlight;
  }
  ```

  This is custom code that redefines border color for header cells. This color is the same as for all other cells. But for the orange theme, the border color for header cells is set to @theme-color-supersuperlight. You can add other custom code the same way.

- If you do not use Less, we recommend writing your custom CSS code in a separate file (e.g. my-flexmonster-styles.css) and to keep the original styles in flexmonster.css without changes. Take a look at the following example: how to add custom CSS for the grid (https://jsfiddle.net/flexmonster/zx4easf4/). It demonstrates how grid colors can be changed via additional CSS.

The full list of examples with custom CSS is available on our examples page (/examples/#custom-css).

What’s next?

You may be interested in the following articles:

- Extended grid customization (/doc/extended-grid-customization/)
- How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/)
- (/doc/localizing-component/)How to add localization (https://www.flexmonster.com/doc/localizing-component/) (/doc/localizing-component/)
- How to define a format for date and time (https://www.flexmonster.com/doc/date-and-time-formatting/)
- (/doc/global-object/)How to set specific options common for all reports (https://www.flexmonster.com/doc/global-object/) (/doc/global-object/)
- How to configure the way that data is exported (https://www.flexmonster.com/doc/export-and-print/)

8.4. Customizing the context menu

Flexmonster provides a context menu for the grid’s and chart’s elements where you can drill through values, open filters, or change aggregations. This tutorial explains how the context menu can be customized to provide new functionality.

The context menu can be customized via the customizeContextMenu (/api/customizecontextmenu/) API call.

Removing items from the context menu

customizeContextMenu returns an array of objects, each of which describes a context menu item. Certain items can be removed by deleting the corresponding object from the array. The following example removes all context menu items:

```
flexmonster.customizeContextMenu(function(items, data, viewType) {
 return [];
});
```

Open the example on JSFiddle (https://jsfiddle.net/flexmonster/q254Lsef/).

Adding new items to the context menu

The following code will add a new item to the flat table context menu:

```
flexmonster.customizeContextMenu(function(items, data, viewType) {
 if (viewType == "flat")
  items.push({
   label: "Switch to charts",
   handler: function() {
    flexmonster.showCharts();
   }
  });
 return items;
});
```

Open the example on JSFiddle (https://jsfiddle.net/flexmonster/1p2qbx2t/).

Check out a full description of the customizeContextMenu parameters (/api/customizecontextmenu/).

What's next?

You may be interested in the following articles:

- How to customize the Toolbar (/doc/customizing-toolbar/)
- How to customize appearance (/doc/customizing-appearance/)
- How to add localization (/doc/localizing-component/)
- How to define a format for date and time (/doc/date-and-time-formatting/)
- How to set specific options common for all reports (/doc/global-object/)
- How to configure the way that data is exported (/doc/export-and-print/)

8.5. Customizing the grid

Besides appearance customization of the component (/doc/customizing-appearance/) in general, Flexmonster also provides functionality for extended grid customization. The customizeCell (/api/customizecell/) API call allows styling up entire columns, rows, or specific cells according to a certain requirement.

About the customizeCell API call

The customizeCell function is triggered for each table cell during rendering, and it has two parameters:

- The cell builder, which contains the cell's current HTML representation and through which the

representation can be customized.
- The Cell Data Object (/api/cell-object/), which contains information about the cell, e.g., its position on the grid and the semantics of data this cell represents.

Check out a full description of the customizeCell parameters (/api/customizecell/).

This guide contains seven examples illustrating how customizeCell helps in achieving visualization goals. To see more examples of customizeCell usage, visit the Examples page (https://www.flexmonster.com/examples/#!customize-cells).

- Alternating row colors (#alternating-rows)
- Styling subtotals and grand totals (#styling-totals)
- Highlighting levels through the entire grid (#highlighting-levels)
- Highlighting cells based on data (#highlighting-by-semantics)
- Styling rows based on conditional formatting (#styling-by-conditional-formatting)
- Representing numbers by icons (#set-numbers-as-icons)
- Adding hyperlinks (#adding-hyperlinks)

Alternating row colors

This sample explains how to highlight every other row on the grid.

Here’s how the customizeCellFunction can be defined:

```
function customizeCellFunction(cell, data) {
  if (data.type == "value") {
    if (data.rowIndex % 2 == 0) {
      cell.addClass("alter1");
    } else {
      cell.addClass("alter2");
    }
  }
}
```

Here is the CSS code sample:

```
#fm-pivot-view .fm-grid-view div.alter1 {
  background-color: #f7f7f7;
}

#fm-pivot-view .fm-grid-view div.alter2 {
  background-color: #fcfcfc;
}
```

In this sample, CSS classes are applied only to cells where data.type is "value", which means that the cell contains data.

Try a live sample on JSFiddle: Alternating row colors (https://jsfiddle.net/flexmonster/4z20ssq4/).

The same approach can be used for alternating column colors.

Styling subtotals and grand totals

Totals in rows and columns can be styled independently. The Cell Data Object (/api/cell-object/) has properties specifying if the cell is subtotal in the compact pivot table, subtotal in the classic form or grand total. Based on the values of these properties in each cell, a CSS class can be added to it. Here is an example:

```
function customizeCellFunction(cell, data) {
  if (data.isGrandTotalRow) cell.addClass("fm-grand-total-r");
}
```

The CSS code sample:

```
#fm-pivot-view .fm-grid-view .fm-grand-total-r {
  background-color: #70C1B3;
}
```

Try it on JSFiddle: Styling subtotals and grand totals (https://jsfiddle.net/flexmonster/Lmx4p4os/).

Highlighting levels through the entire grid

Highlighting entire levels in the pivot table, including data cells, is available through the Cell Data Object (/api/cell-object/)'s level property. In this example, classes with colors for three levels are added to cells depending on their level property.

Note that .fm-level-{n} classes are already assigned to the row header cells, and customizeCellFunction adds classes to cells with data to highlight the entire level.

The CSS code sample:

```
#fm-pivot-view .fm-grid-view div.fm-level-0 {
  background-color: #6869aa;
}

#fm-pivot-view .fm-grid-view div.fm-level-1 {
  background-color: #8e8ebf;
}

#fm-pivot-view .fm-grid-view div.fm-level-2 {
  background-color: #aeaecf;
}
```

The customizeCellFunction can be defined as follows:

```
function customizeCellFunction(cell, data) {
  if (data.level != undefined) {
      cell.addClass("fm-level-"+data.level);
  }
}
```

Live sample on JSFiddle: Highlighting levels through the entire grid (https://jsfiddle.net/flexmonster/v06jx81y/).

Highlighting cells based on their semantics – member, hierarchy, measure

The Cell Data Object (/api/cell-object/) has measure, rows, and columns properties that contain info about the cell's semantics. Take a closer look at several samples demonstrating the usage of these properties.

4.a. Highlighting data about a certain member

This example describes how to highlight cells containing info about Canada regardless of their position:

```
function customizeCellFunction(cell, data) {
  if (data.rows) {
    for (var i = 0; i < data.rows.length; i++) {
      if (data.rows[i]["hierarchyCaption"] == "Country"
          && data.rows[i]["caption"] == "Canada") {
        cell.attr["hierarchy"] = data.rows[i]["hierarchyCaption"];
        cell.attr["member"] = data.rows[i]["caption"];
      }
    }
  }
  if (data.columns) {
    for (var i = 0; i < data.columns.length; i++) {
      if (data.columns[i]["hierarchyCaption"] == "Country"
          && data.columns[i]["caption"] == "Canada") {
        cell.attr["hierarchy"] = data.columns[i]["hierarchyCaption"];
        cell.attr["member"] = data.columns[i]["caption"];
      }
    }
  }
}
```

Then, after the attributes were added, the following CSS selector can be written:

```
#fm-pivot-view .fm-grid-view div[hierarchy="Country"][member="Canada"] {
  background-color: #CCCCCC;
}
```

Try a live sample on JSFiddle: Highlighting Country Canada in any place in rows or columns (https://jsfiddle.net/flexmonster/bbefo4z6/).

4.b. Highlighting the rows with a certain measure

This sample is similar to the previous sample, but information about the measure is added to cells as follows:

```
function customizeCellFunction(cell, data) {
  if (data.measure) {
    cell.attr["measure"] = data.measure.name;
  }
}
```

Each HTML cell will have a measure attribute, and the following CSS selector will highlight the rows with Price:

```
#fm-pivot-view .fm-grid-view div[measure="Price"] {
  background-color: #B2DBBF;
}
```

Live sample on JSFiddle: Highlighting rows with measure Price (https://jsfiddle.net/flexmonster/5mjp55rb/).

4.c. Adding all the semantic info to cell builder attributes

This JSFiddle example demonstrates another way of adding all the semantic info to cell builder attributes: Highlighting cells based on their semantic (https://jsfiddle.net/flexmonster/1354qwh6/).

This sample takes into account the position of the hierarchies in rows and columns.

Styling rows based on conditional formatting

It is also possible to highlight the entire row in the pivot table if the condition of the conditional formatting is true for at least one cell in this row.

The Cell Data Object (/api/cell-object/) has the conditions property containing a list of the conditional formatting ids that are true for this cell.

Check the JSFiddle example that applies formatting to the entire row if the condition is true for at least one cell in this row: Styling rows based on the conditional formatting (https://jsfiddle.net/flexmonster/nonte3qv/).

Representing numbers by icons

This example demonstrates how to replace cell values with the images depending on which interval the value belongs to: high, middle, and so forth. Try it on JSFiddle: Custom cells demo (https://jsfiddle.net/flexmonster/6shdmx9u/).

Adding hyperlinks

This case describes adding links to all the countries in the report. It can be done through the text property of the cell builder. Try a live sample on JSFiddle: Add hyperlinks (https://jsfiddle.net/flexmonster/q1gtwj48/).

The possibilities of customizeCell are not limited by the above seven cases. This API call covers many variants of visualization. To learn more about the customizeCell function and the Cell Data Object, see the API documentation:

- customizeCell (/api/customizecell/)
- Cell Data Object (/api/cell-object/)

What’s next?

You may be interested in the following articles:

- How to customize the Toolbar (/doc/customizing-toolbar/)
- How to customize the appearance (/doc/customizing-appearance/)
- How to customize the context menu (/doc/customizing-context-menu/)

8.6. Customizing the pivot charts

Flexmonster Pivot Charts can be customized and adjusted to the user’s needs. The customizeChartElement (https://www.flexmonster.com/api/customizechartelement/) API call allows setting colors for chart elements, adding custom attributes, and so on.

This guide contains the following sections:

- About the customizeChartElement method (#about-customizeChartElement)
- Using customizeChartElement (#using-customizeChartElement)
- Changing default colors for charts (#changing-colors)

About the customizeChartElement method

The customizeChartElement method is triggered for each chart element during the rendering. As a parameter, it takes customizeChartElementFunction which has two parameters:

- Either the HTML (https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement) or SVG (https://developer.mozilla.org/en-US/docs/Web/API/SVGElement) element, which contains the current representation of the chart element and through which the element’s representation can be customized.
- The Chart Data Object (https://www.flexmonster.com/api/chart-data-object/) or the Chart Legend Data Object (https://www.flexmonster.com/api/chart-legend-data-object/) containing information about the chart element.

To learn more about the customizeChartElement API call and its parameters, refer to the documentation (https://www.flexmonster.com/api/customizechartelement/).

Using customizeChartElement

This example illustrates how to highlight the chart elements containing info about the United States regardless of their position:

1. Create a function to check whether the data contains the given member:

```
function contains(data, memberName) {
  if (data && data.rows) {
    for (let i = 0; i < data.rows.length; i++) {
      if (data.rows[i].uniqueName == memberName) {
        return true;
      }
    }
  }
  if (data && data.columns) {
    for (let i = 0; i < data.columns.length; i++) {
      if (data.columns[i].uniqueName == memberName) {
        return true;
      }
    }
  }
  return false;
}
```

2. By default, all the hierarchy’s child members have their own color shade when the chart is expanded. We will create a function to prepare a custom shade for each child member of the United States. This function will retrieve a lightness of the element’s default color, apply it to the custom color, and return a resulting shade:

   ```
   function applyShade(newColor, oldColor) {
       /* Get the lightness of the default color */
       /* Here we divide it by 2 to adjust the default lightness
       and get a darker color as a result */
       let colorLightness = tinycolor(oldColor).toHsl().l/2;

       /* Apply the lightness to the new color */
       let result = newColor.toHsl();
       result.l = colorLightness;

       /* Return the created color shade */
       return tinycolor(result);
   }
   ```

   This function uses the TinyColor library (https://github.com/bgrins/TinyColor) for color conversions, so include the library into your project with the following script:

   ```
   <script
    src="https://cdnjs.cloudflare.com/ajax/libs/tinycolor/1.4.1/tinycolor.min.js"
   >
   </script>
   ```

3. If the data contains the United States member, color the corresponding chart element with a certain shade of purple:

   ```
   function customizeChartElementFunction(element, data) {
     let newColor = tinycolor("#1a0019");
     if (contains(data, "country.[united states]")) {
       if (data.chartType == "pie") {
         if (element.querySelector('path')) {
           element.querySelector('path').style.fill = applyShade(newColor,
           element.querySelector('path').style.fill).toRgbString();
         }
       }
       else {
         element.style.fill = applyShade(newColor, element.style.fill)
   ```

```
          .toRgbString();
      }
    }
  }
```

The pie chart should be handled separately as it has a different structure of chart elements.

4. Update the chart legend item’s color:

```
function customizeChartElementFunction(element, data) {
  // code from step 2
  if (data && data.type == 'legend') {
    if (data.member && data.member.uniqueName == "country.[united states]"
    && !data.isExpanded) {
      element.querySelector(".fm-icon-display").style
      .backgroundColor = applyShade(newColor, data.color).toHexString();
    }
    if (data.tuple && isChild(data.tuple, data.member.uniqueName,
    "country.[united states]") && !data.isExpanded) {
      element.querySelector(".fm-icon-display").style
      .backgroundColor = applyShade(newColor, data.color).toHexString();
    }
  }
}
```

The isChild function checks whether the current member is a child of the given member (in our case, of the United States member). Its definition is the following:

```
function isChild(tuple, member, parentMember) {
    let i = 0;
    while (tuple[i].uniqueName != member) {
        if (tuple[i].uniqueName == parentMember) {
            return true;
        }
        i++;
    }
    return false;
}
```

See the full example on JSFiddle (https://jsfiddle.net/flexmonster/0t8gv2cp/).

Changing default colors for charts

One more useful example shows how to change the default colors for charts, check it out on JSFiddle (https://jsfiddle.net/flexmonster/wLqupkc5/). We use .fm-charts-color-n to set the color for the nth chart sector. In this example, we specify six colors. The seventh sector is specified with fill: none. This trick is used so that custom colors are repeated if there are more than six chart sectors. If the seventh sector was not specified, Flexmonster would use its own colors.

What’s next?

You may be interested in the following articles:

- How to customize the grid (https://www.flexmonster.com/doc/customizing-grid/)
- How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/)
- How to customize the appearance (https://www.flexmonster.com/doc/customizing-appearance/)
- How to customize the context menu (https://www.flexmonster.com/doc/customizing-context-menu/)

8.7. Localizing the component

On our website, you can see an example of a localized version of Flexmonster Pivot in this demo: Localization (/demos/localization). The component can be localized to Spanish, French, Italian, Portuguese, Mandarin, etc. The following article describes how the component can be localized.

The default language for all text elements in the pivot table is English. The full process of component localization involves two steps:

1. Create a localization JSON file (#create-localization-file)
2. Set the localization for the pivot table (#set-localization)

Step 1. Create your localization JSON file

The localization JSON file is a JSON file that has a list of all texts and labels that are used in Flexmonster Pivot and can be localized.

If you plan to use one of the localizations used in our demos, you are free to download the respective localization file from our site. Just click the file with language below or right click and save locally. Another option is to use a localization file from our CDN (#from-cdn).

- English (https://github.com/flexmonster/pivot-localizations/blob/master/en.json)
- German (https://github.com/flexmonster/pivot-localizations/blob/master/de.json)
- Español (https://github.com/flexmonster/pivot-localizations/blob/master/es.json)
- Français (https://github.com/flexmonster/pivot-localizations/blob/master/fr.json)
- Italiano (https://github.com/flexmonster/pivot-localizations/blob/master/it.json)
- Português (https://github.com/flexmonster/pivot-localizations/blob/master/pt.json)
- Chinese (https://github.com/flexmonster/pivot-localizations/blob/master/zh.json)
- Ukrainian (https://github.com/flexmonster/pivot-localizations/blob/master/uk.json)
- Magyar (https://github.com/flexmonster/pivot-localizations/blob/master/hu.json)
- Dutch (https://github.com/flexmonster/pivot-localizations/blob/master/nl.json)
- Turkish (https://github.com/flexmonster/pivot-localizations/blob/master/tr.json)

If you have downloaded one of the above files, you can jump to Step 2. Set localization for Flexmonster Pivot (#set-localization).

If you want to use a different language, it is possible to create your own localization file. We recommend using the English localization file as a template. Download it (https://github.com/flexmonster/pivot-localizations/blob/master/en.json), make a copy, and replace all English texts with your own. After creating a new localization file, our team suggests sharing it with the community to help other users. New localization files can be added by creating a new pull request on GitHub (https://github.com/flexmonster/pivot-localizations). The Flexmonster team will highly appreciate any new localization files and add them to our website.

In the following example, we localize the Calculated Value window in the pivot table to French.

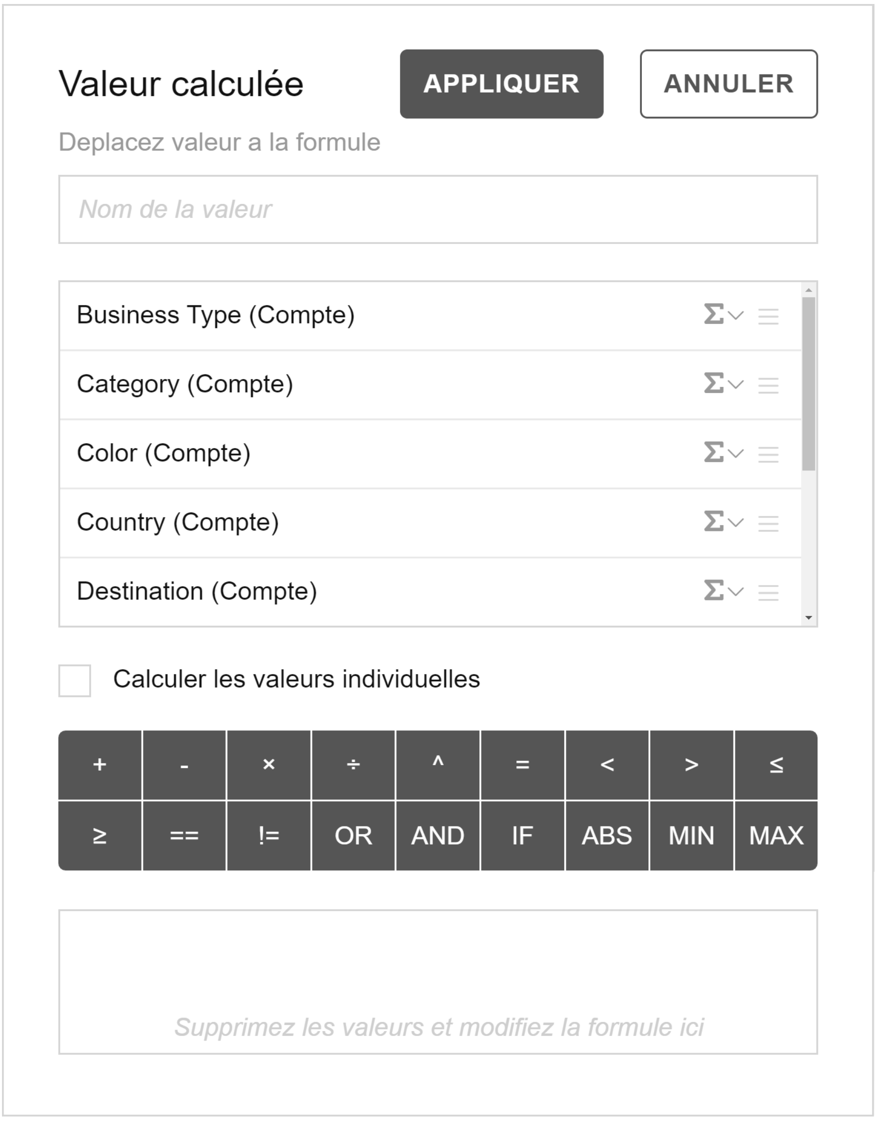
The original section of the localization file looks as follows.

```
"calculatedView": {
  "measureBox": "Drag Values To Formula",
  "measureName": "Value name",
  "calculateIndividualValues": "Calculate individual values"
}
```

Now we replace the respective terms with French.

```
"calculatedView": {
  "measureBox": "Deplacez valeur a la formule",
  "measureName": "Nom de la valeur",
  "calculateIndividualValues": "Calculer les valeurs individuelles"
}
```

Once the localization JSON file is loaded (see the next step), your pivot table should show the following changes in the Calculated Value view:



Your localization can be partial. For example, if you don’t need to localize the pivot table completely, you can replace only those JSON objects that you want to translate. If certain label translations are not mentioned in the localization JSON file, they will be set to their default English values.

Step 2. Set localization for Flexmonster Pivot

Localization can be set as inline JSON or a URL to the localization JSON file. To preset localization for all reports, set the localization property in the global object. Here is an example:

URL to JSON

```
new Flexmonster({
    container: "pivotContainer",
    global: {
        localization: "loc/fr.json"
    },
    ...
});
```

JSON

```
new Flexmonster({
    container: "pivotContainer",
    global: {
        localization: {
            "calculatedView": {
                "measureBox": "Deplacez valeur a la formule",
                "measureName": "Nom de la valeur",
                "formula": "Formule"
            }
        }
    },
    ...
});
```

To specify different localization for a particular report, set the localization property in the report object. Here is an example:

URL to JSON

```
new Flexmonster({
    container: "pivotContainer",
    report: {
        localization: "loc/fr.json"
    },
    ...
});
```

JSON

```
new Flexmonster({
    container: "pivotContainer",
    report: {
       localization: {
           "calculatedView": {
                "measureBox": "Deplacez valeur a la formule",
                "measureName": "Nom de la valeur",
                "formula": "Formule"
            }
        }
    },
    ...
});
```

Use a localization file from our CDN

Instead of downloading a localization JSON file, you can localize the component with a file from our CDN. Just specify a URL to the needed localization file:

For all reports

```
new Flexmonster({>
    container: "pivotContainer",
    global: {
        localization: "https://cdn.flexmonster.com/loc/fr.json"
    },
    ...
});
```

For a particular report

```
new Flexmonster({
    container: "pivotContainer",
    report: {
        localization: "https://cdn.flexmonster.com/loc/fr.json"
    },
    ...
});
```

Find a link to the file with your localization:

- English: https://cdn.flexmonster.com/loc/en.json
- German: https://cdn.flexmonster.com/loc/de.json
- Español: https://cdn.flexmonster.com/loc/es.json
- Français: https://cdn.flexmonster.com/loc/fr.json
- Italiano: https://cdn.flexmonster.com/loc/it.json
- Português: https://cdn.flexmonster.com/loc/pt.json
- Chinese: https://cdn.flexmonster.com/loc/zh.json
- Ukrainian: https://cdn.flexmonster.com/loc/uk.json
- Magyar: https://cdn.flexmonster.com/loc/hu.json
- Dutch: https://cdn.flexmonster.com/loc/nl.json
- Turkish: https://cdn.flexmonster.com/loc/tr.json

What’s next?

You may be interested in the following articles:

- (/doc/customizing-toolbar/)How to customize the Toolbar (https://www.flexmonster.com/doc/customizing-toolbar/) (/doc/customizing-toolbar/)
- How to customize appearance (/doc/customizing-appearance/)
- How to define a format for date and time (/doc/date-and-time-formatting/)
- How to set specific options common for all reports (/doc/global-object/)
- How to configure the way that data is exported (/doc/export-and-print/)

9. Updating to the latest version

9.1. Updating to the latest version

This guide describes how to update Flexmonster to the latest version. If you don’t have Flexmonster yet, get a free trial here (https://www.flexmonster.com/download-page/).

Table of contents:

Updating Flexmonster Pivot

We recommend updating Flexmonster Pivot Table & Charts to the latest version with Flexmonster CLI (https://www.flexmonster.com/doc/cli-overview). If needed, install the CLI globally using npm:

```
npm install -g flexmonster-cli
```

To update Flexmonster without npm, follow this guide (#update-flexmonster-zip).

Run the appropriate command from the project folder to update Flexmonster in your application:

Pure JavaScript

```
flexmonster update flexmonster
```

Angular

```
flexmonster update ng-flexmonster
```

React

```
flexmonster update react-flexmonster
```

Vue

```
flexmonster update vue-flexmonster
```

If you are updating from a previous major version to 2.8, check out the Migration guide from 2.7 to 2.8 (https://www.flexmonster.com/doc/migration-guide-from-2-7-to-2-8/).

To ensure that you updated the component, you can check the **component's version** by clicking on the grid and pressing Ctrl + Alt + i (Option + Control + i on macOS). The information about the version of the component and its edition will be shown in the pop-up window.

You're done! You are ready to use the most recent version of Flexmonster Pivot Table & Charts with its newest features.

Note: Being our active customer (during the first year after purchase or renewal payment) you are able to **update for free**. If you are our client and you would like to renew the maintenance, please contact our team (https://www.flexmonster.com/contact) for more details.

Updating Flexmonster CLI

Update Flexmonster CLI to the latest version with the following npm command:

```
npm update -g flexmonster-cli
```

Updating Flexmonster Accelerator

Step 1. Uninstall the Accelerator.

Step 2. Get the most recent version of the Accelerator with the following CLI command:

```
flexmonster update accelerator
```

The flexmonster update accelerator command will do the following:

- Download the .zip archive with Flexmonster Accelerator.
- Unpack the files in the current folder.
- Automatically initiate the installation using the Flexmonster Accelerator.msi setup file.

When the installation begins, just follow the wizard.

Updating Flexmonster Data Server

Step 1. Stop the Data Server by closing the program.

Step 2. Get the most recent version of the Data Server with the update command. Execute it from the folder with the Data Server:

```
flexmonster update fds
```

Updating will not override flexmonster-config.json, log files, or any additional file or folder.

Step 3. Start Flexmonster Data Server by launching the executable file.

Updating Flexmonster by downloading a package

Updating Flexmonster Pivot

Step 1. Go to the Client’s area (https://www.flexmonster.com/client/) and download the latest version of Flexmonster Pivot Table for JavaScript there.

Step 2. Replace your flexmonster/ folder with the new one.

If you are updating from a previous major version to 2.8, check out the Migration guide from 2.7 to 2.8 (https://www.flexmonster.com/doc/migration-guide-from-2-7-to-2-8/).

To ensure that you updated the component, you can check the **component's version** by clicking on the grid and pressing Ctrl+Alt+i. The information about the version of the component and its edition will be shown in the pop-up window.

Note: Being our active customer (during the first year after purchase or renewal payment) you are able to **update for free**. If you are our client and you would like to renew the maintenance, please contact our team (https://www.flexmonster.com/contact) for more details.

Updating Flexmonster Accelerator

Step 1. Uninstall the Accelerator.

Step 2. Go to the Client's area (https://www.flexmonster.com/client/) and download the latest version of Flexmonster Accelerator for SSAS there.

Step 3. Install the Accelerator using the updated MSI installer from the package.

Updating Flexmonster Data Server

Step 1. Stop the Data Server by closing the program.

Step 2. Go to the Client's area (https://www.flexmonster.com/client/) and download the most recent version of Flexmonster Data Server for your operating system.

Step 3. Replace the Data Server file with the updated version from the downloaded package.

Step 4. Start the Data Server by launching the executable file.

9.2. Release notes

9.3. Migration guide from 2.7 to 2.8

From Flexmonster Pivot Table & Charts 2.7 to 2.8

Follow this tutorial for a comfortable and quick migration to the new major version. Find more details below:

1. New features (#new)
    - New data source: custom data source (#custom-api)
    - Field List in the drill-through view for Elasticsearch (#elastic-drillthrough)
    - New database connector: MongoDB (#mongo-connector)
    - New integrations: Vue.js, React Native, R, Python (#frameworks)
    - New aggregation: percentage of Parent Row/Column Total (#new-aggs)
    - New events (#new-events)
2. Updates (#updates)
    - Updates in element IDs (#id-updates)
    - Updates in supported formats (#formats)
    - Updates in the Flexmonster module (#fm-module)
    - Updates in API calls (#api-upd)
3. Updating from previous versions (#prev-guides)

New features

New data source: custom data source

The custom data source API enables fetching already aggregated data from any server to Flexmonster Pivot. Develop a powerful back end with your logic, process the data right on the server, and show the result with Flexmonster's custom data source API. Learn more here (/doc/introduction-to-custom-data-source-api/).

Field List in the drill-through view for Elasticsearch

In the drill-through view for the Elasticsearch data source, it is now possible to select fields to display the chosen details about the value. To open the Field List, click on the arrow in the upper right corner of the table in the drill-through view.

New database connector: MongoDB

This connector greatly simplifies connecting to MongoDB. Use it instead of the Data Compressor for Node.js. Refer to our documentation (/doc/mongodb-connector/) to learn more about connecting to MongoDB.

New integrations: Vue.js, React Native, R, Python

Flexmonster is now available for Vue.js and React Native frameworks, R-powered applications with Shiny, and Python-based applications including Jupyter Notebook. Find more details about these integrations in the following articles:

- Vue.js (https://www.flexmonster.com/doc/integration-with-vue/)
- React Native (https://www.flexmonster.com/doc/integration-with-react-native/)
- R Shiny (https://www.flexmonster.com/doc/integration-with-r-shiny/)
- Python (Django) (https://www.flexmonster.com/doc/integration-with-django/)
- Python (Jupyter Notebook) (https://www.flexmonster.com/doc/integration-with-jupyter-notebook/)

New aggregation: percentage of Parent Row/Column Total

In version 2.8, a new aggregation was added that shows a percentage of parent row/column total aggregation is now available in the aggregation list.

New events

Two new events were added:

- drillthroughopen – String. Triggered when the drill-through view is opened. Learn more here (https://www.flexmonster.com/api/drillthroughopen/).
- drillthroughclose – String. Triggered when the drill-through view is closed. Learn more here (https://www.flexmonster.com/api/drillthroughclose/).

Updates

Updates in element IDs

In version 2.8.16, a number of element IDs were changed to class names. It was done to ensure that there are no duplicate IDs on the page.

If you have custom CSS, update your CSS selectors according to the list below (e.g., change the #fm-add-btn selector to .fm-add-btn).

Here is a full list of IDs changed to class names:

- #fm-add-btn
- #fm-add-group-view
- #fm-aggr-display
- #fm-aggregations-view
- #fm-alert-view
- #fm-and-label
- #fm-branding-bar
- #fm-btn-add-measure
- #fm-btn-add-measure-2
- #fm-btn-close-fields
- #fm-btn-collapse-expand-all
- #fm-btn-connect
- #fm-btn-open-fields
- #fm-build-version
- #fm-calc-display
- #fm-calculated-view
- #fm-cancel-btn
- #fm-chart
- #fm-chart-legend
- #fm-charts-filters-btn
- #fm-charts-filters-container
- #fm-charts-view
- #fm-cols-filter
- #fm-cols-resize
- #fm-cols-sheet
- #fm-conditions
- #fm-conditions-dropdown
- #fm-data-sheet
- #fm-datepicker-1
- #fm-datepicker-2
- #fm-dates-filter-view
- #fm-details-label
- #fm-drag-handle"
- #fm-drillthrough-view
- #fm-fields-view
- #fm-filter-label
- #fm-filter-sort-row
- #fm-filter-view
- #fm-filters-col
- #fm-font-family
- #fm-font-size
- #fm-formula-input
- #fm-func-btn-group
- #fm-grid-view
- #fm-header-toolbar
- #fm-icon-display
- #fm-info-icon
- #fm-inp-proxy-url
- #fm-interval-dropdown
- #fm-labels-filter-btn
- #fm-labels-filter-view
- #fm-landscape-radio
- #fm-left-scroll-button

- #fm-link
- #fm-list-wrapper
- #fm-lst-columns
- #fm-lst-hierarchies
- #fm-lst-measures
- #fm-lst-pages
- #fm-lst-rows
- #fm-measures-dropdown
- #fm-members-filter-list
- #fm-message-label
- #fm-moreicon-display
- #fm-name-input
- #fm-next-btn
- #fm-num-input-1
- #fm-num-input-2
- #fm-numbers-filter-view
- #fm-page-filter
- #fm-periods-dropdown
- #fm-popUp-modal-overlay
- #fm-popup-conditional
- #fm-popup-format-cells
- #fm-popup-olap
- #fm-popup-options
- #fm-portrait-radio
- #fm-preloader-view
- #fm-prev-btn
- #fm-prompt-view
- #fm-remove-btn
- #fm-right-scroll-button
- #fm-rows-filter
- #fm-rows-resize
- #fm-rows-sheet
- #fm-sample
- #fm-select-counter
- #fm-sheet-headers
- #fm-sort-col
- #fm-sort-label
- #fm-spinner
- #fm-text-display
- #fm-time-filter-view
- #fm-txt-input-1
- #fm-txt-input-2
- #fm-ui-dp-month
- #fm-ui-dp-year
- #fm-values
- #fm-values-filter-view
- #fm-version-label
- #fm-wrap-columns
- #fm-wrap-measures
- #fm-wrap-pages
- #fm-wrap-rows

For instance, the #fm-grid-view selector in the following CSS code:

```
#fm-pivot-view #fm-grid-view div.alter1 {
  background-color: #f7f7f7;
}
```

should be changed to .fm-grid-view:

```
#fm-pivot-view .fm-grid-view div.alter1 {
  background-color: #f7f7f7;
}
```

Updates in supported formats

The following updates were made concerning the supported formats:

- The OCSV-1 format is no longer supported in version 2.8.
- When connecting to a local CSV file, the CSV data is now saved to the report in JSON format, so it is possible to open the report on another host.

Updates in the Flexmonster module

In version 2.8, the following dependencies were removed from the flexmonster/lib/ folder:

- file.min.js
- jszip.min.js
- zlib.min.js
- elasticsearch.min.js

Also, the type definition file d.ts is now a part of the Flexmonster module.

Updates in API calls

- getCell().measure.availableAggregations now contains the array of all available aggregations instead of being empty when all the default aggregations are supported.
- report.slice.measures[n].availableAggregations are now saved only if they were defined in a report.
- The aggregations property was added to the Mapping object to define available aggregations. Starting from version 2.8, the Slice object's availableAggregations property, which has the same purpose, is considered deprecated. Use the Mapping object with the aggregations property instead.

Updating from previous versions

If migrating from a previous major version, follow these tutorials:

- Migration guide from 2.6 to 2.7 (https://www.flexmonster.com/doc/migration-guide-from-2-6-to-2-7/)
- Migration guide from 2.5 to 2.6 (https://www.flexmonster.com/doc/migration-guide-from-2-5-to-2-6/)
- Migration guide from 2.4 to 2.5 (https://www.flexmonster.com/doc/migration-guide-from-2-4-to-2-5/)
- Migration guide from 2.3 to 2.4 (https://www.flexmonster.com/doc/migration-guide-from-2-3-to-2-4/)
- Migration guide from 2.2 to 2.3 (https://www.flexmonster.com/doc/migration-guide/)

9.4. Migration guide from 2.6 to 2.7

From Flexmonster Pivot Table & Charts 2.6 to 2.7

Follow this tutorial for a comfortable and quick migration to the new major version. Find more details below:

1. Removed API calls (#removed)
2. Updated API calls (#updated)
3. Changes to the report object (#report)
4. New data source: Elasticsearch (#elasticsearch)
5. Localization files update (#localization)
6. Changes to relative URLs (#urls)
7. Updating from the previous versions (#previous)

Removed API calls

1. getFilterProperties was removed in version 2.7. Use getFilter() (https://www.flexmonster.com/api/getFilter/) instead.
2. setTopX was removed in version 2.7. Use setFilter() (https://www.flexmonster.com/api/setFilter/) instead.
3. setBottomX was removed in version 2.7. Use setFilter() (https://www.flexmonster.com/api/setFilter/) instead.

Updated API calls

1. getFilter(hierarchyName: String) now returns a Filter Object (https://www.flexmonster.com/api/filter-object/) (instead of an array with members).
2. setFilter(hierarchyName: String, filter: Filter Object) now takes a Filter Object (https://www.flexmonster.com/api/filter-object/) as a second argument (instead of an array with members).

Changes to the report object

1. columns.filter, rows.filter, and reportFilters.filter from the Slice Object (https://www.flexmonster.com/api/slice-object/) were changed, find the new structure here: Filter Object (https://www.flexmonster.com/api/filter-object/). The following filtering properties were removed (they remain available in terms of backward compatibility):
    - filter.negation was removed in version 2.7. Use exclude from the Filter Object (https://www.flexmonster.com/api/filter-object/) instead.
    - filter.quantity was removed in version 2.7. Use query from the Filter Object (https://www.flexmonster.com/api/filter-object/) instead.
    - filter.type was removed in version 2.7. Use query from the Filter Object (https://www.flexmonster.com/api/filter-object/) instead.
2. Options Object (/api/options-object/) was extended with the following properties:
    - filter – Object. Filtering options:
        - timezoneOffset – Number. The difference (in minutes) between UTC and the user's local time zone. Used to adjust the dates in the filter. *Default value: user's local time*.
        - weekOffset – Number. The number of days to be added to the start of the week (Sunday). Used to adjust the first day of the week in the filter's calendar. *Default value: 1 (Monday is the first day of the week).*
        - dateFormat – String. The date pattern to format dates in the filter's date inputs. Has two possible values: "dd/MM/yyyy" and "MM/dd/yyyy". *Default value: "dd/MM/yyyy"*.
3. dataSource from the Report Object (https://www.flexmonster.com/api/report-object/) was extended with the following properties:
    - host (optional) – String | Object. The host for the connection (only for the "elasticsearch" data

source type). Can be set either as a URL string ("host": "https://olap.flexmonster.com:9200") or as a Host Object (https://www.elastic.co/guide/en/elasticsearch/client/javascript-api/current/host-reference.html).
    - index (optional) – String. The name of the Elasticsearch index to connect to (only for the "elasticsearch" data source type).
    - mapping (optional) – Object. An additional setting to configure index mapping (only for "elasticsearch" data source type). Use the name of the field as a key of the object with the following parameters:
        - caption (optional) – String. Overrides the default name of the field.
        - visible (optional) – Boolean. When set to false, it hides the field from the Field List.
        - interval (optional) – String. Used for the date histogram (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html). Check out the supported time units (https://www.elastic.co/guide/en/elasticsearch/reference/current/common-options.html#time-units).
        - time_zone (optional) – String. Used for the date histogram (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html). You can specify time zones either as an ISO 8601 UTC offset (e.g. +01:00 or -08:00) or as a time zone ID as specified in the IANA time zone database, such as America/Los_Angeles. Check out this example (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html#_timezone).
        - format (optional) – String. Used for the date histogram (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html). Check out the date format/pattern (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-daterange-aggregation.html#date-format-pattern).

New data source: Elasticsearch

Flexmonster now supports connecting to an Elasticsearch data source out of the box. Find more details in our blog post (https://www.flexmonster.com/blog/introducing-flexmonster-pivot-for-elasticsearch/).

Localization files update

New localization labels were added in version 2.7. Get the updated localization files from our GitHub repository (https://github.com/flexmonster/pivot-localizations).

Changes to relative URLs

In 2.7 we added support of ./ (that means the current directory) for dataSource.filename and localization URLs. All details are covered below:

- If the URL is without / or ./ – the path will be resolved using the componentFolder + filename.
- If the URL starts with / – the path will be resolved using the root directory (website root).
- If the URL starts with ./ – the path will be resolved using the current directory (current HTML page path).
- If the URL starts with ../ – the path will be resolved using the parent folder of the componentFolder.

Updating from the previous versions

If migrating from a previous major version, follow these tutorials:

- Migration guide from 2.5 to 2.6 (/doc/migration-guide-from-2-5-to-2-6/)
- Migration guide from 2.4 to 2.5 (/doc/migration-guide-from-2-4-to-2-5/)
- Migration guide from 2.3 to 2.4 (/doc/migration-guide-from-2-3-to-2-4/)
- Migration guide from 2.2 to 2.3 (/doc/migration-guide/)

9.5. Migration guide from 2.5 to 2.6

From Flexmonster Pivot Table & Charts 2.5 to 2.6

Our priority is to develop cool new features and to provide easy migration between versions of Flexmonster. To update to version 2.6 just follow our guide on updating to the latest version (https://www.flexmonster.com/doc/updating-to-the-latest-version/). In most cases, all existing code is fully compatible with the new version.

The full list of new features is available in our release notes for version 2.6 (https://www.flexmonster.com/release-notes/version-2-6-0/). There you will also find links to our tutorials explaining how to use the new features.

XML deprecation

The majority of our customers have already switched to using a JSON format for the reports, so the XML format is being deprecated from version 2.6. Old XML reports can be converted to JSON via an online XML to JSON converter (https://www.flexmonster.com/convert-xml-report/) or an npm package converter (https://www.npmjs.com/package/pivot-xml-report-converter).

Changes in the report object and API calls

Among other improvements, we have added the ability to select a measure multiple times allowing you to apply different aggregations on the same measure. For example, a revenue measure can be selected 2 times for a pivot table showing the aggregated values revenue (sum) and revenue (average). Check out the JSFiddle demo (https://jsfiddle.net/flexmonster/g5m73ezp/).

In version 2.5 and earlier, the measure’s unique name was used to apply sorting, filtering on values, and table sizes. In version 2.6, since one measure can be selected multiple times, it may be necessary to provide not only the unique name of the measure but also its aggregation type to identify the measure. Thus, measure (String) is replaced with measure (Object). measure has the following properties:

- uniqueName – String. The measure’s unique name.
- aggregation (optional) – String. The measure’s aggregation type.

The aggregation parameter is optional, it only needs to be specified when the report contains a measure with multiple aggregations.

Example

setTopX API call usage in version 2.5:

```
flexmonster.setTopX("Category", 2, "Price");
```

setTopX API call usage in version 2.6:

```
flexmonster.setTopX("Category", 2, {"uniqueName": "Price"});
```

"Price" is replaced with {"uniqueName": "Price"}.

All changes are backward compatible, so code from version 2.5 will still remain functional. Moreover, if a report composed in version 2.5 is loaded into version 2.6 and then saved, the changes to the report object will be applied automatically. However, if your own code composes/parses reports or uses API calls from the list below, it is recommended to update the code manually by replacing measure (String) with measure (Object). Below is a list of places where this update is necessary.

To update in the report object:

- report.slice.sorting.row.measure
- report.slice.sorting.column.measure
- report.slice.rows[n].filter.measure
- report.slice.columns[n].filter.measure
- report.slice.expands.rows[n].measure
- report.slice.expands.columns[n].measure
- report.slice.drills.rows[n].measure
- report.slice.drills.columns[n].measure
- report.options.chart.activeMeasure
- report.tableSizes.rows[n].measure
- report.tableSizes.columns[n].measure

Check out the updated structure of the Report Object (https://www.flexmonster.com/api/report-object/).

To update in API calls usage:

- getFilterProperties (/api/getfilterproperties/) return has changed.
- setBottomX (/api/setBottomX/) input parameter has changed.
- setTopX (/api/setTopX/) input parameter has changed.
- sortValues (/api/sortValues/) input parameter has changed.

Updating from previous versions

When migrating from previous major versions, follow these tutorials:

- Migration guide from 2.4 to 2.5 (/doc/migration-guide-from-2-4-to-2-5/)
- Migration guide from 2.3 to 2.4 (/doc/migration-guide-from-2-3-to-2-4/)
- Migration guide from 2.2 to 2.3 (/doc/migration-guide/)

9.6. Migration guide from 2.4 to 2.5

From Flexmonster Pivot Table & Charts 2.4 to 2.5

You can start using our new major version quickly and easily. To migrate from version 2.4, just update the Flexmonster component (https://www.flexmonster.com/doc/updating-to-the-latest-version/). You don’t need to make any changes to your existing code or custom styles! Version 2.5 is fully compatible with code that worked in 2.4. API methods are the same, and the use of the pivot table is unchanged.

Our main improvement is reducing memory usage by around 50%. Now you can analyze more data at once and process it quicker.

One more update is the new dark theme. To add the dark theme to your project, just include this reference:

```
<link rel="stylesheet" type="text/css" href="/theme/dark/flexmonster.min.css" />
```

To find the full list of updates in version 2.5, check out the release notes for version 2.5 (https://www.flexmonster.com/release-notes/version-2-5-0/).

Note if you are migrating from version 2.3 or 2.2, we suggest reading the following tutorials first:

- Migration guide from 2.3 to 2.4 (/doc/migration-guide-from-2-3-to-2-4/)
- Migration guide from 2.2 to 2.3 (/doc/migration-guide/)

9.7. Migration guide from 2.3 to 2.4

From Flexmonster Pivot Table & Charts 2.3 to 2.4

Read this tutorial for comfortable and quick migration to the new major version. The tutorial contains the following sections:

1. How to embed the component into your website (#embedPivotComponent)
2. API updates (#updates)
3. Built-in themes (#themes)
4. (#css)Changes in CSS (#css) (#css)
5. (#toolbar)Changes in the Toolbar customization (#toolbar) (#toolbar)
6. Accelerators (#accelerator)
7. (#docs)Documentation for older versions (#docs) (#docs)

How to embed the component into your website

Starting from version 2.4, jQuery is no longer required. Before embedding the component, include flexmonster.js:

```
<script src="flexmonster/flexmonster.js"></script>
```

In previous versions the $.flexmonster() and embedPivotComponent() functions were used to embed the component. In version 2.4, they were deprecated. However, they are still supported for backward compatibility. Note, to use $.flexmonster() you need to include the jQuery library before flexmonster.js.

To embed Flexmonster in version 2.4, create the pivot table object like this:

```
var pivot = new Flexmonster({
    container:String,
    componentFolder:String,
    global:Object,
    width:Number,
    height:Number,
    report:Object|String,
    toolbar:Boolean,
    customizeCell:Function,
    licenseKey:String
})
```

Notice that the container property should specify a selector for the HTML element that you would like to use as a container for the component. This is the only required property in the configuration object. If you run new Flexmonster({container: "pivotContainer"}), where "pivotContainer" is the selector of the container HTML element, – an empty component without the Toolbar will be added with the default width and height. The configuration object can have the following properties:

- container – String. The selector of the HTML element that will be a container for the component.
- componentFolder – String. The URL of the component's folder that contains all the necessary files. It is also used as the base URL for report files, localization files, styles, and images. *Default value: "flexmonster/"*.
- global – Report Object (https://www.flexmonster.com/api/report-object/). Allows you to preset options for all reports. These options can be overridden for specific reports. The structure of this object is the same as for report.
- width – Number | String. The width of the component on the page (either in pixels or as a percentage). *Default value: "100%"*.
- height – Number | String. The height of the component on the page (either in pixels or as a percentage). *Default value: 500*.
- report – Report Object (https://www.flexmonster.com/api/report-object/) | String. The property to set a report. It can be an inline report JSON or a URL to the report JSON. XML reports are also supported for backward compatibility.
- toolbar – Boolean. The parameter to embed the Toolbar (the one that is shown in the pivot table demo (https://www.flexmonster.com/demos)) or not. *Default value: false*.
- customizeCell – Function. Allows the customization of separate cells.
- licenseKey – String. Your license key.

Event handlers can also be set as properties for the Flexmonster object.

The initialization statement returns a reference to the created component instance. The API methods are available through this reference.

API updates

Please have a look at the changes in our API:

1. The customizeCell hook was improved:
    - The html property (string containing HTML) was replaced with a special cell builder cell. It is an object that contains the current representation of the cell on the grid and through which the cell representation can be customized. It has the following properties and methods:
        - attr – Object. All attributes and their values for the HTML element. Custom attributes can be added to the cell and, for example, used in CSS selectors to identify the cell. Read more info about CSS attribute selectors here (https://css-tricks.com/attribute-selectors/).
        - classes – Array of strings. The array of classes assigned to the cell. The addClass() method should be used to add new classes.
        - style – Object. A CSS object of the element that will be put in the style attribute of the element for inline styling.
        - tag – String. The tag of the element (each cell has a tag: "div").
        - text – String. The text of the element. It may also contain HTML, for example, icons to expand, collapse, drill up and down, sorting, etc.
        - addClass(value: String) – Method. Use this method to add new classes to the element.
        - toHtml() – Method. Returns the HTML string that represents the cell. It gathers all the properties of the cell builder object into HTML. This is how the cell gets added to the grid.
    - The data property, which contains information about the cell, now provides more information. Check out the full list of properties in the Cell Data Object (https://www.flexmonster.com/api/cell-object/).

Note that customizeCell is NOT backward compatible. You need to update your customizeCell functions in accordance with these changes.

2. The Conditional Format Object (https://www.flexmonster.com/api/conditional-format-object/) was updated:
    - The formula property property was simplified, now it should contain only the condition (e.g. '#value > 100').
    - The trueStyle property was renamed to format. Also, inside the format object, the fontSize property should be set as "14px" instead of "14".
    - The falseStyle property was removed.

   For example, this definition of the condition object:

```
var condition = {
 id: 1,
 formula: 'if(#value > 400000, "trueStyle")',
 trueStyle: {fontSize : 10, backgroundColor: "#33BB33"}
};
```

   should be replaced with the following one:

```
var condition = {
 id: 1,
 formula: '#value > 400000',
 format: {fontSize : "10px", backgroundColor: "#33BB33"}
};
```

   There is also a new property formatCSS in the Conditional Format Object (https://www.flexmonster.com/api/conditional-format-object/). This is a string which represents the ready to use CSS string of the format style object (formerly trueStyle).
   The syntax from the previous version is functional in terms of backward compatibility.

3. The fullscreen API call was removed. Use Fullscreen from the Toolbar instead.
4. The getPages API call was renamed to getReportFilters.
5. The property pages from the Report Object (https://www.flexmonster.com/api/report-object/) was renamed to reportFilters (with backward compatibility).
6. The properties configuratorMatchHeight and pagesFilterLayout were removed from the Options Object (https://www.flexmonster.com/api/options-object/).
7. The default value of configuratorActive property inside the Options Object (https://www.flexmonster.com/api/options-object/) was changed to false.
8. Two new events were added:
    - beforegriddraw – String. Triggered before grid rendering.
    - aftergriddraw – String. Triggered after grid rendering.

Built-in themes

Starting from version 2.4, you can choose between several predefined CSS themes. All themes are placed inside the theme/ folder. To set a theme, add the CSS reference. For example, to show the lightblue theme, insert the following line of code:

```
<link rel="stylesheet" type="tex
t/css" href="/theme/lightblue/flexmonster.min.css" />
```

Our 2.3-styled theme is available as well and can be set like this:

```
<link rel="stylesheet" type="text/css" href="/theme/old/flexmonster.min.css" />
```

Changes in CSS

In version 2.4 we decided to simplify the maintenance of CSS. The following changes were made:

- flexmonster.css and flexmonster.toolbar.css were merged into one file flexmonster.css. Note that the changes with CSS files are NOT backward compatible, so you need to replace your old flexmonster.css file. Also note that selectors were changed, so you need to update any custom CSS files based on flexmonster.css.
- We started using Less (http://lesscss.org/) (a CSS pre-processor). With the download package you get CSS, minified CSS, and the Less source code files. It allows you to add your own styles on top of Less files and then compile them to CSS. This is extremely useful for creating your own themes. Just copy the Less file with one of our themes and redefine several variables with basic colors to create your custom theme.

Changes in the Toolbar customization

The order of tabs in the Toolbar was changed. Additionally, the icons are now a vector. If you like the new icons, you can use them in your customizeToolbar function (a JS function for the beforetoolbarcreated event) like this:

```
// add new tab
tabs.unshift({
  id: "fm-tab-save",
  title: "Save",
  handler: "savetabHandler",
  icon: this.icons.save
});

tabs.unshift({
  id: "fm-tab-reload",
  title: "Reload",
  handler: "reloadCube",
  icon: this.icons.connect
});
```

You can use the icon property to add your own icons. This property has the type string and represents an HTML tag containing the custom icon for the new tab. Note, in this case any CSS with old icons should be removed.

Accelerators

Starting from version 2.4, our Accelerators work considerably faster. If you are using Flexmonster Accelerator for

Microsoft Analysis Services or Flexmonster Accelerator for Pentaho Mondrian, keep in mind that they should be updated as well.

Documentation for older versions

If you need documentation from some older versions, it is always available at the following links:

- Documentation 2.3 (https://s3.amazonaws.com/flexmonster/2.3/docs/Flexmonster-Documentation-2.3.pdf)
- API Reference 2.3 (https://s3.amazonaws.com/flexmonster/2.3/docs/Flexmonster-API-Reference-2.3.pdf)
- Documentation 2.2 (https://s3.amazonaws.com/flexmonster/2.2/docs/Flexmonster-Documentation-2.2.pdf)
- API Reference 2.2 (https://s3.amazonaws.com/flexmonster/2.2/docs/Flexmonster-API-Reference-2.2.pdf)

9.8. Migration guide from 2.2 to 2.3

From Flexmonster Pivot Table & Charts 2.2 to 2.3

The new version of Flexmonster offers several new methods. Additionally, some existing methods were improved and some were removed. This guide will help you migrate from the previous version to the new one.

To migrate from the previous major version to 2.3, read the following:

1. License keys (/doc/migration-guide/#keys)
2. (/doc/migration-guide/#embedPivotComponent)How to embed Flexmonster into your website (https://www.flexmonster.com/doc/migration-guide/#embedPivotComponent) (/doc/migration-guide/#embedPivotComponent)
3. (/doc/migration-guide/#updates)API updates (https://www.flexmonster.com/doc/migration-guide/#updates) (/doc/migration-guide/#updates)
4. New methods (/doc/migration-guide/#added)
5. Removed methods (/doc/migration-guide/#removed)
6. The new structure of the report object (https://www.flexmonster.com/doc/migration-guide/#structure)
7. Event list (/doc/migration-guide/#eventList)
8. (/doc/migration-guide/#toolbarLocalization)Localization of the Toolbar (https://www.flexmonster.com/doc/migration-guide/#toolbarLocalization) (/doc/migration-guide/#toolbarLocalization)
9. Folder structure (https://www.flexmonster.com/doc/migration-guide/#folderStructure)
10. Documentation for version 2.2 (https://www.flexmonster.com/doc/migration-guide/#docs)

License keys

Version 2.3 requires new license keys. If you have an active maintenance – contact our customer service team and you will receive new **development** and **production license keys** for free. To renew annual maintenance or for further details please contact (http://www.flexmonster.com/contact/) our sales team.

How to embed Flexmonster into your website

Version 2.3 is based on jQuery. In order for the component scripts to work as expected, make sure you include a reference to the jQuery library in the document before the scripts. It should look like this:

```
<script src="flexmonster/lib/jquery.min.js"></script>
<script src="flexmonster/flexmonster.js"></script>
```

In the previous version, the embedPivotComponent() function was used to embed the component. Now, in version 2.3 it has been deprecated, but will be supported in this version for backward compatibility. Instead of embedPivotComponent(), use a jQuery call like this:

```
var pivot = $("#pivotContainer").flexmonster({
    componentFolder:String,
    global:Object,
    width:Number,
    height:Number,
    report:Object|String,
    toolbar:Boolean,
    licenseKey:String})
```

As a parameter, the jQuery call takes a selector of the HTML element that you would like to have as a container for the component.

All the properties are optional in the configuration object. If you run $("#pivotContainer").flexmonster() – an empty component without the Toolbar will be added with the default width and height. The configuration object passed to the jQuery call can have the following properties:

- componentFolder – String. The URL of the component's folder that contains all the necessary files. It is also used as the base URL for report files, localization files, styles, and images. *Default value: "flexmonster/"*.
- global – Report Object (https://www.flexmonster.com/api/report-object/). Allows you to preset options for all reports. These options can be overridden for specific reports. The structure of the object is the same as for report.
- width – Number | String. The width of the component on the page (either in pixels or as a percentage). *Default value: "100%"*.
- height – Number | String. The height of the component on the page (either in pixels or as a percentage). *Default value: 500*.
- report – Report Object (https://www.flexmonster.com/api/report-object/) | String. The property to set a report. It can be an inline report JSON or a URL to the report JSON. XML reports are also supported for backward compatibility. In the previous version this property was called configUrl. The structure of the report object has also changed. The chapter New structure of the report object (https://www.flexmonster.com/doc/migration-guide/#structure) explains how to use the new structure.
- toolbar – Boolean. The parameter to embed the Toolbar (the one that is shown in the pivot table demo (https://www.flexmonster.com/demos)) or not. In the previous version this property was called withToolbar. *Default value: false*
- licenseKey – String. Your license key.

Event handlers can also be set as properties for the jQuery call. Find the list here (https://www.flexmonster.com/doc/migration-guide/#eventList).

The jQuery initialization statement returns a reference to the created component instance. The API methods are available through this reference.

After initialization, you can obtain an instance reference to the component by using a selector like this: var pivot = $("#pivot").data("flexmonster");

API updates

Here is a list of methods that were changed:

1. save
   In the previous version, reports were saved in an XML format. Now save returns the report as a JSON object. See the New structure of the report object (https://www.flexmonster.com/doc/migration-guide/#structure) for further details.
2. getAllMeasures and getMeasures
   The calculated property was removed from the measure object. To see if a measure was calculated, check the formula property of the measure object. It defines whether the measure is calculated or not.
3. getOptions and setOptions
   Some of the component's options were changed. If some option was renamed, its name must be changed anywhere it was used. For example:

   ```
   var options = flexmonster.getOptions();
   options.showTotals = false;
   flexmonster.setOptions(options);
   flexmonster.refresh();
   ```

   In version 2.3 showTotals was renamed to grid.showTotals, so change

   ```
   options.showTotals = false;
   ```

   to

   ```
   options.grid.showTotals = false;
   ```

   The same needs to be done for all the renamed options. All references to removed options should be deleted. See the renamed options among the other changes to the report object here (https://www.flexmonster.com/doc/migration-guide/#structure).
4. setHandler
   setHandler was replaced with on() and off(). The event list was extended. The full list can be found here (https://www.flexmonster.com/doc/migration-guide/#eventList).
5. getReport and setReport
   The object structure was changed. The chapter New structure of the report object (https://www.flexmonster.com/doc/migration-guide/#structure) explains how to use the new structure.

New methods

These methods were added to version 2.3:

1. dispose() – prepares the pivot table instance to be deleted by the browser's garbage collection.
2. getData(options: Object, callbackHandler: Function, updateHandler: Function) – enables integration with 3rd party charting libraries (FusionCharts, Highcharts, Google Charts, etc). Returns the data from the pivot table component. The options object can have the slice property. If slice in options is not defined, the API call will return the data displayed in the pivot table.
3. off(eventName: String, functionName: String) – removes the event listener. If the functionName property is not defined, all event listeners will be removed.
4. updateData(params: Object | Array) – the addJSON and connectTo API calls were replaced by this API call. It is used to update the data for the report without clearing the report.

More detailed information about each API call is available in the API Reference (https://www.flexmonster.com/api/).

Removed methods

1. addMeasure should be removed. It can be replaced with addCalculatedMeasure(measure)

(https://www.flexmonster.com/api/addcalculatedmeasure/).

2. addDataArray, addDataRecord, addDimension, and addHierarchy should be removed. A JSON data source should be used instead. The first element of the JSON object is used to define data types, captions, group fields under one dimension, and hierarchies.

```
var jsonData = [
    {
        "Color":{ "type": "string" },
        "M":{
            "type": "month",
            "dimensionUniqueName": "Days", // addDimension analog
            "dimensionCaption": "Days",
            "caption":"Month"
        },
        "W":{
            "type": "weekday",
            "dimensionUniqueName": "Days",
            "dimensionCaption": "Days",
            "caption":"Week Day"
        },
        "Country":{
            "type":"level",
            // addHierarchy analog
            "hierarchy": "Geography",
            "level":"Country"
        },
        "State":{
            "type":"level",
            "hierarchy": "Geography",
            "level":"State",
            "parent": "Country"
        },
        "City":{
            "type":"level",
            "hierarchy": "Geography",
            "parent": "State"
        },
        "Price":0,
        "Quantity":{ "type": "number" }
    },
    {
        // addDataRecord and addDataArray analog
        "Color":"green",
        "M":"September",
        "W":"Wed",
        "Country" : "Canada",
        "State" : "Ontario",
        "City" : "Toronto",
        "Price":174,
        "Quantity":22
   },
   {
        "Color":"red",
        "M":"March",
        "W":"Mon",
        "Time":"1000",
        "Country" : "USA",
        "State" : "California",
        "City" : "Los Angeles",
        "Price":1664,
        "Quantity":19
    }
];
```

3. addStyleToMember
4. addUrlToMember
5. getColumnWidth
6. getLabels
7. getRowHeight
8. getValue should be removed. It can be replaced with getCell() (https://www.flexmonster.com/api/getcell/).

   ```
   flexmonster.getCell(1,1);
   ```

9. gridColumnCount
10. gridRowCount
11. percentZoom
12. removeAllMeasures should be removed. It can be replaced with removeAllCalculatedMeasures() (https://www.flexmonster.com/api/removeallcalculatedmeasures/).
13. removeMeasure should be removed. It can be replaced with removeCalculatedMeasure(name) (https://www.flexmonster.com/api/removecalculatedmeasure/).
14. setChartTitle should be removed. It can be replaced with

    ```
    flexmonster.setOptions({chart:{title: "Chart One"}})
    ```

15. setColumnWidth
16. setGridTitle should be removed. It can be replaced with

    ```
    flexmonster.setOptions({ grid:{title: "Table One"}})
    ```

17. setLabels should be removed. It can be replaced in one of two ways:
    - Using global.localization inside the global object:

      ```
      var global = {
          localization: "loc-en.json",
          dataSource: {
              dataSourceType: "csv", filename: "data.csv"
          }
      };
      var pivot = $("#pivotContainer").flexmonster({
          global: global,
          licenseKey: "XXX"
      });
      ```

    - While setting a report with setReport() (/api/setreport/):

      ```
      var report = {
          localization: "loc-en.json",
          dataSource: {
              dataSourceType: "csv",
              filename: "data.csv"
      ```

```
            }
        };
        flexmonster.setReport(report);
```

18. setRowHeight
19. setSelectedCell
20. setStyle
21. zoomTo

The new structure of the report object

In version 2.3 the report object was restructured. Properties were logically grouped into sub-objects. For example, data sources related properties are located inside the dataSource sub-object now.

JSON report objects from the previous version and XML reports are fully supported in terms of backward compatibility. This means that you can open or set reports from the previous major version. Note that all reports will now be saved in the new JSON structure, so the component can be used to migrate reports to the new version. We also created a small utility (https://www.flexmonster.com/blog/moving-to-json-converter-for-old-xml-reports/) for converting XML reports to JSON. This is an easy and convenient way of migrating to the new format.

Here is an example of the new report object:

```
{
    "dataSource": {
        "dataSourceType": "microsoft analysis services",
        "proxyUrl": "http://olap.flexmonster.com/olap/msmdpump.dll",
        "dataSourceInfo": "WIN-IA9HPVD1RU5",
        "catalog": "Adventure Works DW Standard Edition",
        "cube": "Adventure Works",
        "binary": false
    },
    "slice": {
        "rows": [
            {
                "uniqueName": "[Geography].[Geography]",
                "filter": {
                    "members": [
                        "[Geography].[Geography].[City].&[Malabar]&[NSW]",
                        "[Geography].[Geography].[City].&[Lavender Bay]&[NSW]",
                        "[Geography].[Geography].[City].&[Matraville]&[NSW]",
                        "[Geography].[Geography].[City].&[Milsons Point]&[NSW]"
                    ],
                    "negation": true
                },
                "sort": "asc"
            },
            {
                "uniqueName": "[Sales Channel].[Sales Channel]",
                "sort": "asc"
            }
        ],
        "columns": [
            {
                "uniqueName": "[Measures]"
            }
        ],
        "measures": [
            {
                "uniqueName": "[Measures].[Reseller Order Count]",
                "aggregation": "none",
                "active": true,
                "format": "29mvnel3"
            }
        ],
        "expands": {
            "expandAll": false,
            "rows": [
                {
                    "tuple": [
                        "[Geography].[Geography].[Country].&[Australia]"
                    ]
                },
                {
                    "tuple": [
                        "[Geography].[Geography].[City].&[Lane Cove]&[NSW]"
                    ]
                }
            ]
        },
        "drills": {
            "drillAll": false,
            "rows": [
                {
                    "tuple": [
                        "[Geography].[Geography].[Country].&[Australia]"
                    ]
                },
                {
                    "tuple": [
                        "[Geography].[Geography].[State-Province].&[NSW]&[AU]"
                    ]
                },
                {
                    "tuple": [
                        "[Geography].[Geography].[City].&[Darlinghurst]&[NSW]"
                    ]
                }
            ]
        }
    },
    "options": {
        "viewType": "grid",
        "grid": {
            "type": "compact",
            "title": "",
            "showFilter": true,
            "showHeaders": true,
            "fitGridlines": false,
            "showTotals": true,
            "showGrandTotals": "on",
            "showExtraTotalLabels": false,
            "showHierarchies": true,
            "showHierarchyCaptions": true,
            "showReportFiltersArea": true,
            "pagesFilterLayout": "horizontal"
        },
        "chart": {
            "type": "bar",
            "title": "",
            "showFilter": true,
            "multipleMeasures": false,
            "oneLevel": false,
            "autoRange": false,
            "reversedAxes": false,
            "showLegendButton": false,
            "showAllLabels": false,
            "showMeasures": true,
            "showOneMeasureSelection": true,
            "showWarning": true,
            "activeMeasure": ""
        },
        "configuratorActive": true,
        "configuratorButton": true,
        "configuratorMatchHeight": false,
        "showAggregations": true,
        "showCalculatedValuesButton": true,
        "editing": false,
        "drillThrough": true,
        "showDrillThroughConfigurator": true,
        "sorting": "on",
        "datePattern": "dd/MM/yyyy",
        "dateTimePattern": "dd/MM/yyyy HH:mm:ss",
        "saveAllFormats": false,
        "showDefaultSlice": true,
        "showEmptyData": false,
        "defaultHierarchySortName": "asc",
        "selectEmptyCells": true,
        "showOutdatedDataAlert": false
    },
    "conditions": [
        {
            "formula": "if(#value < 40, 'trueStyle')",
            "trueStyle": {
                "backgroundColor": "#FFCCFF",
                "color": "#000033",
                "fontFamily": "Arial",
                "fontSize": 12
            },
            "falseStyle": {}
        }
    ],
    "formats": [
        {
            "name": "29mvnel3",
            "thousandsSeparator": " ",
            "decimalSeparator": ".",
            "decimalPlaces": -1,
            "maxDecimalPlaces": -1,
            "maxSymbols": 20,
            "currencySymbol": "$",
            "currencySymbolAlign": "left",
            "nullValue": "",
            "infinityValue": "Infinity",
            "divideByZeroValue": "Infinity",
            "textAlign": "right",
            "isPercent": false
        }
    ],
    "tableSizes": {
        "columns": [
            {
                "tuple": [],
                "measure": "[Measures].[Reseller Order Count]",
                "width": 182
            }
        ]
    },
    "localization": "loc-ua.json"
}
```

Here is a table that illustrates how the properties of the Report Object (https://www.flexmonster.com/api/report-object/) were changed between versions 2.2 and 2.3:

Version 2.2	Version 2.3
browseForFile	dataSource.browseForFile
chartActiveMeasure	options.chart.activeMeasure
chartActiveMeasures	options.chart.activeMeasures
chartActiveTupleIndex	options.chart.activeTupleIndex
chartAutoRange	options.chart.autoRange
chartMultipleMeasures	options.chart.multipleMeasures
chartOneLevel	options.chart.oneLevel
chartOneLevelTuple	options.chart.oneLevelTuple
chartPosition	options.chart.position
chartReversedAxes	options.chart.reversedAxes
chartTitle	options.chart.title
chartType	options.chart.type
classicView	grid.type
columnHeaderSizes	tableSizes.columns
columnSizes	tableSizes.columns
columnSorting	slice.sorting.column
columns	slice.columns
columns.customSorting	slice.columns.sortOrder
columns.sortName	slice.columns.sort
conditions	conditions
configuratorActive	options.configuratorActive
configuratorButton	options.configuratorButton
configuratorMatchHeight	options.configuratorMatchHeight

customData	dataSource.customData
customFields	customFields
customSort	slice.sortOrder
dataSourceType	dataSource.dataSourceType
datePattern	options.datePattern
dateTimePattern	options.dateTimePattern
defaultHierarchySortName	options.defaultHierarchySortName
drillAll	slice.drills.drillAll
drillThrough	options.drillThrough
drilledColumns	slice.drills.columns
drilledRows	slice.drills.rows
editing	options.editing
effectiveUserName	dataSource.effectiveUserName
expandAll	slice.expands.expandAll
expandedColumns	slice.expands.columns
expandedRows	slice.expands.rows
fieldSeparator	dataSource.fieldSeparator
filename	dataSource.filename
fitGridlines	options.grid.fitGridLines
flatOrder	slice.flatOrder
flatView	options.grid.type
formats	formats
gridTitle	options.grid.title
ignoreQuotedLineBreaks	dataSource.ignoreQuotedLineBreaks
labels	labels
localSettingsUrl	localization
localeIdentifier	dataSource.localeIdentifier
maxChunkSize	removed
measures	slice.measures
memberProperties	slice.memberProperties
pages	slice.pages
pages.customSorting	slice.pages.sortOrder
pages.sortName	slice.pages.sort
password	dataSource.password
roles	dataSource.roles
rowFilterSizes	tableSizes.rows
rowHeaderSizes	tableSizes.rows
rowSizes	tableSizes.rows
rowSorting	slice.sorting.row
rows	slice.rows
rows.customSorting	slice.rows.sortOrder
rows.sortName	slice.rows.sort
saveAllFormats	options.saveAllFormats
showAggregations	options.showAggregations
showAllChartLabels	options.chart.showAllLabels
showCalculatedValuesButton	options.showCalculatedValuesButton
showChartLegendButton	options.chart.showLegendButton
showChartMeasures	options.chart.showMeasures
showChartOneMeasureSelection	options.chart.showOneMeasureSelection
showChartsWarning	options.chart.showWarning
showDefaultSlice	options.showDefaultSlice
showDrillThroughConfigurator	options.showDrillThroughConfigurator
showEmptyData	options.showEmptyData
showFilter	options.grid.showFilter
showFilterInCharts	options.chart.showFilter

showFiltersExcel	removed
showGrandTotals	options.grid.showGrandTotals
showHeaders	options.grid.showHeaders
showHierarchies	options.grid.showHierarchies
showHierarchyCaptions	options.grid.showHierarchyCaptions
showMemberProperties	options.showMemberProperties
showOutdatedDataAlert	options.showOutdatedDataAlert
showReportFiltersArea	options.grid.showReportFiltersArea
showTotals	options.grid.showTotals
sorting	options.sorting
useOlapFormatting	options.useOlapFormatting
username	dataSource.username
viewType	options.viewType
zoom	removed
Also, these new options were added:	
	options.grid.pagesFilterLayout
	options.selectEmptyCells

Note:

- showFiltersExcel was removed from options. Instead, use the showFilter parameter from export options inside the exportTo() API call, as follows:

  ```
  flexmonster.exportTo("excel", {showFilters: true})
  ```

  Or

  ```
  flexmonster.exportTo("excel", {showFilters: false})
  ```

  In exportTo() showFilters is false by default.

- drillAll was removed from options and moved to slice inside the Report Object (https://www.flexmonster.com/api/report-object/).
- expandAll was removed from options and moved to slice inside the Report Object (https://www.flexmonster.com/api/report-object/).
- ignoreQuotedLineBreaks was removed from options and moved to dataSource inside the Report Object (https://www.flexmonster.com/api/report-object/).

  ```
  //To stop ignoring line breaks in quotes
  var report = flexmonster.getReport();
  report.dataSource.ignoreQuotedLineBreaks = false;
  flexmonster.setReport(report);
  ```

Event list

List of renamed events:

- jsPivotCreationCompleteHandler was renamed to reportcomplete – String. Triggered when the OLAP structure or data from the report, localization file, and all data source files are loaded successfully and the

grid/chart is rendered. This event means the operations can be performed with the component.
- jsCellClickHandler was renamed to cellclick – String. Triggered when a cell is clicked on the grid.
- jsFieldsListOpenHandler was renamed to fieldslistopen – String. Triggered when the built-in Field List is opened.
- jsFieldsListCloseHandler was renamed to fieldslistclose – String. Triggered when the built-in Field List is closed.
- jsFilterOpenHandler was renamed to filteropen – String. Triggered when a filter icon is pressed.
- jsFullScreenHandler was removed.
- jsReportChangeHandler was renamed to reportchange – String. Triggered when a report is changed in the component.
- jsReportLoadedHandler was renamed to reportcomplete – String. Triggered when the component finishes loading a report file.
- jsPivotUpdateHandler was renamed to update – String. Triggered when the component loaded data, updated data slice, filter or sort. In other words, when a change occurred in the component.

List of new events:

- celldoubleclick – String. Triggered when a cell is clicked on the grid twice.
- dataerror – String. Triggered when some error occurred during the loading of data. Please use olapstructureerror for OLAP data sources.
- datafilecancelled – String. Triggered when the Open file dialog was opened and a user clicks the Cancel button. It happens when:
    1. A user clicks menu Connect -> To local CSV/JSON.
    2. The API method updateData() (https://www.flexmonster.com/api/updateData/) is called with the parameter browseForFile: true.
- dataloaded – String. Triggered when the component finishes loading data. Please use olapstructureloaded for OLAP data sources.
- loadingdata – String. Triggered when data starts loading from local or remote CSV or JSON. Please use loadingolapstructure for OLAP data sources.
- loadinglocalization – String. Triggered when the localization file starts loading.
- loadingolapstructure – String. Triggered when the structure of an OLAP cube starts loading.
- loadingreportfile – String. Triggered when a report file starts loading.
- localizationerror – String. Triggered when an error appears while loading a localization file.
- localizationloaded – String. Triggered when a localization file finishes loading.
- olapstructureerror – String. Triggered when an error appears while loading an OLAP structure.
- olapstructureloaded – String. Triggered when an OLAP structure finishes loading.
- openingreportfile – String. Triggered when:
    1. A user clicks Open -> Local report.
    2. The API method open() (https://www.flexmonster.com/api/open/) is called.
- querycomplete – String. Triggered after a data query is complete.
- queryerror – String. Triggered if an error occurs while running the query.
- ready – String. Triggered when the component’s initial configuration is complete and the component is ready to receive API calls.
- reportfilecancelled – String. Triggered when a user clicks the menu Open -> Local report and clicks the Cancel button.
- reportfileerror – String. Triggered when an error occurs during the loading of the report file.
- runningquery – String. Triggered before a data query is started. It is used for both cases when data is already loaded and stored in the component’s local storage or when data is loaded from external data storage in the case of an OLAP data source. A data query is started when:
    1. A slice is changed.
    2. Any filter is used.
    3. A drill-up or a drill-down query is performed.
    4. Columns or rows are expanded.
    5. A drill through is used.

Localization of the Toolbar

In the previous version, the Toolbar could be localized by passing translations to the toolbarLocalization parameter in embedPivotComponent(). Now localization of the Toolbar is set inside the localization file. To set localization for only the Toolbar, open your localization file, find the following code:

```
 "toolbar": {
    "connect":
…
}
```

Replace the respective terms with your language. Save the file and reload the component. For more details on how to set localization for the pivot table, please read Localizing the component (https://www.flexmonster.com/doc/localizing-component).

Folder structure

FOLDERS	CONTENTS
assets/	Contains the theme images
lib/	Includes additional JavaScript libraries
toolbar/	Contains the Toolbar files: images, JS, and CSS
flexmonster.css	The theme CSS
flexmonster.js	The main Flexmonster JavaScript file

Note that the folder structure was changed. Now html5-assets/ is replaced with assets/, lib/ and flexmonster.css.

Documentation for version 2.2

If you are looking for documentation from version 2.2, it is available here: Documentation 2.2 (https://s3.amazonaws.com/flexmonster/2.2/docs/Flexmonster-Documentation-2.2.pdf) and API Reference 2.2 (https://s3.amazonaws.com/flexmonster/2.2/docs/Flexmonster-API-Reference-2.2.pdf).

10. Flexmonster CLI Reference

10.1. Overview

Flexmonster CLI is a command-line interface tool for Flexmonster.

The CLI can:

- Download Flexmonster Pivot or its framework wrappers.
- Create sample projects with Flexmonster.
- Get Flexmonster Data Server or Flexmonster Accelerator.
- Update all of the above.

This guide contains the following sections:

1. Flexmonster CLI installation (#installation)

Flexmonster CLI installation

To install Flexmonster CLI, you will need Node.js and npm. Get it here (https://docs.npmjs.com/downloading-and-installing-node-js-and-npm) if it's not already installed on your machine.

Then install Flexmonster CLI globally to use its commands on your computer:

```
npm install -g flexmonster-cli
```

Now a new flexmonster command is available in the console. If needed, see troubleshooting (https://www.flexmonster.com/doc/troubleshooting-cli/).

CLI command syntax

flexmonster command syntax:

```
flexmonster commandNameOrAlias requiredArg optionalArg [options]
```

For example:

```
flexmonster create react es6 --install
```

This command will create a simple React + ES6 project with Flexmonster and install all the npm dependencies. The command consists of the following parts:

- flexmonster — the executable used to run all the CLI commands.
- create — the command name.
- react — the required argument.
- es6 — the required argument.
- --install (or -i) — the option.

Option names have the -- prefix (e.g., --install), and option aliases are prefixed with - (e.g, -i). Command arguments are not prefixed (e.g., react).

Available commands

Flexmonster CLI has the following commands:

Command	Alias	Description
create	c	Creates a sample project with Flexmonster based on a chosen language and framework
add	a	Downloads Flexmonster Pivot, framework wrappers, Flexmonster Data Server, Theme Builder, or Flexmonster Accelerator
update	u	Updates Flexmonster Pivot, framework wrappers, Flexmonster Data Server, or Flexmonster Accelerator
version	v	Outputs the Flexmonster CLI version and notifies about CLI updates if any.
help	h	Provides help on the available commands and options

Basic usage

This section will help you get started with the CLI. If you have any issues when running the CLI commands, visit our troubleshooting page (https://www.flexmonster.com/doc/troubleshooting-cli/).

See Flexmonster CLI version

To see the CLI version, run the following command:

```
flexmonster version
```

Get help on commands

Get the list of all the available commands and options with the help command:

```
flexmonster help
```

To see the description, options, and usage examples for a particular CLI command, run the following command:

```
flexmonster commandNameOrAlias --help
```

Create a sample project with Flexmonster

To create and run a sample JavaScript project with Flexmonster, use the following CLI command:

```
flexmonster create javascript -r
```

The create command also allows setting up sample projects based on TypeScript or different front-end frameworks. See more details about the create command (/doc/flexmonster-create/).

Add Flexmonster to a project

Install the flexmonster npm module with the add command. Run this command from the project folder containing package.json:

```
flexmonster add flexmonster
```

The add command can also download Flexmonster Data Server, Flexmonster Accelerator, or framework wrappers. Learn more about the add command in the documentation (/doc/flexmonster-add/).

Update Flexmonster

Update Flexmonster Pivot with the update command:

```
flexmonster update flexmonster
```

Framework wrappers and our additional tools (the Data Server and the Accelerator) can also be updated with the update command. See the full description of this command (/doc/flexmonster-update).

Troubleshooting

If you have any issues when running the CLI commands, visit our troubleshooting page (https://www.flexmonster.com/doc/troubleshooting-cli/).

What’s next?

You may be interested in the following articles:

- flexmonster create (/doc/flexmonster-create/)
- flexmonster add (https://www.flexmonster.com/doc/flexmonster-add/)
- flexmonster update (https://www.flexmonster.com/doc/flexmonster-update/)
- flexmonster version (https://www.flexmonster.com/doc/flexmonster-version/)
- flexmonster help (/doc/flexmonster-help/)

10.2. Troubleshooting the CLI

This section provides solutions to the errors you might encounter while working with Flexmonster CLI. If you cannot find your error here, post a question to our Help Forum (https://www.flexmonster.com/forum/).

Console error: ‘flexmonster : File C:\Users\User\AppData\Roaming\npm\flexmonster.ps1 cannot be loaded because running scripts is disabled on this system.’

This error means that your current Windows PowerShell execution policy does not allow scripts. To run the CLI commands, try another command prompt or set a new PowerShell execution policy with this command:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Learn more about execution policies in Microsoft documentation (https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-7).

10.3. flexmonster create

flexmonster create|c *[requiredArg] [options]*

The Flexmonster CLI command that creates a sample project with Flexmonster based on a chosen language or front-end framework. Also can install the needed npm dependencies and run the project.

Arguments

The flexmonster create command can be run with the following required arguments:

Argument	Description
angular	Creates a sample Angular project with Flexmonster Pivot. For example: `flexmonster create angular -r`

react	Creates a sample React project with Flexmonster Pivot. You can choose either ES6 or TypeScript.
	
	• React + ES6:
	`flexmonster create react es6 -r`
	
	• React + TypeScript:
	`flexmonster create react typescript -r`
	
	If the language is not specified, you will be prompted to choose it during the initialization process.

vue	Creates a sample Vue project with Flexmonster Pivot. For example: `flexmonster create vue -r`

typescript	Creates a sample project with Flexmonster Pivot based on TypeScript. For example:
	
	`flexmonster create typescript -r`
	
	If needed, Webpack can be added to the project:
	
	`flexmonster create typescript webpack -r`

javascript	Creates a sample project with Flexmonster Pivot based on pure JavaScript. For example: `flexmonster create javascript -r`
electron	Creates a sample ElectronJS project with Flexmonster Pivot. For example:

```
flexmonster create electron -r
```

Options

The flexmonster create command can be run with the following options:

Option	Description
--install, -i	Automatically installs all the needed npm dependencies.
--run, -r	Automatically installs all the needed npm dependencies and runs the project.
--help, -h	Shows the short guidance on this command in the console. Use this command to see all the possible uses of the create command.

What's next?

You may be interested in the following articles:

- flexmonster add (https://www.flexmonster.com/doc/flexmonster-add/)
- flexmonster update (https://www.flexmonster.com/doc/flexmonster-update/)
- flexmonster version (https://www.flexmonster.com/doc/flexmonster-version/)
- flexmonster help (https://www.flexmonster.com/doc/flexmonster-help/)

10.4. flexmonster add

flexmonster add|a [requiredArg] [options]

The Flexmonster CLI command that downloads the needed Flexmonster tool to the project.

Arguments

The flexmonster add command can be run with the following required arguments:

Argument	Description
accelerator	Downloads the package with Flexmonster Accelerator for SSAS (the Flexmonster Accelerator.msi file) to the current folder. For example: `flexmonster add accelerator -i`
fds	Downloads Flexmonster Data Server with the sample configuration and the sample data to the current folder. For example: `flexmonster add fds -r`
flexmonster	Downloads Flexmonster Pivot to the node_modules/ folder and adds it as an npm dependency to the package.json file. Run the command from the folder containing package.json. For example: `flexmonster add flexmonster`
ng-flexmonster	Downloads the Flexmonster Angular module to the node_modules/ folder and adds it as an npm dependency to the package.json file. Run the command

	from the folder containing package.json. For example: `flexmonster add ng-flexmonster`
react-flexmonster	Downloads the Flexmonster React module to the node_modules/ folder and adds it as an npm dependency to the package.json file. Run the command from the folder containing package.json. For example: `flexmonster add react-flexmonster`
vue-flexmonster	Downloads the Flexmonster Vue module to the node_modules/ folder and adds it as an npm dependency to the package.json file. Run the command from the folder containing package.json. For example: `flexmonster add vue-flexmonster`
theme-builder	Downloads the custom theme builder (https://github.com/flexmonster/custom-theme-builder) to the current folder. For example: `flexmonster add theme-builder -i`

Options

The flexmonster add command can be run with the following options:

Option	Description
--install, -i	Depending on the tool added, the option can automatically: • Initiate the installation of Flexmonster Accelerator for SSAS. • Install all the needed npm dependencies for the custom theme builder.
--run, -r	Automatically runs Flexmonster Data Server after the download.
--help, -h	Shows the short guidance on this command in the console. Use this command to see all the possible uses of the add command.

What’s next?

You may be interested in the following articles:

- flexmonster create (https://www.flexmonster.com/doc/flexmonster-create/)
- flexmonster update (https://www.flexmonster.com/doc/flexmonster-update/)
- flexmonster version (https://www.flexmonster.com/doc/flexmonster-version/)
- flexmonster help (https://www.flexmonster.com/doc/flexmonster-help/)

10.5. flexmonster update

flexmonster update|u [requiredArg] [options]

The Flexmonster CLI command that updates Flexmonster Pivot, Flexmonster wrapper for a chosen framework, Flexmonster Data Server, or Flexmonster Accelerator.

Arguments

The flexmonster update command can be run with the following required arguments:

Argument	Description
accelerator	Updates Flexmonster Accelerator for SSAS. The command can be executed from any folder. For example: `flexmonster update accelerator`
fds	Updates Flexmonster Data Server. Execute the command from the folder with the Data Server. If flexmonster update is run from the folder containing the flexmonster-config.json file, the flexmonster update fds command will be executed automatically. Updating will not override flexmonster-config.json, log files, or any additional file or folder.
flexmonster	Updates Flexmonster Pivot. Execute the command from the project folder. If flexmonster update is run from the folder containing package.json with the "flexmonster" npm dependency, the flexmonster update flexmonster command will be executed automatically.
ng-flexmonster	Updates the Flexmonster Angular module. Execute the command from the project folder. If flexmonster update is run from the folder containing package.json with the "ng-flexmonster" npm dependency, the flexmonster update ng-flexmonster command will be executed automatically.
react-flexmonster	

10.6. flexmonster version

flexmonster version|v [options]

The Flexmonster CLI command that outputs the CLI version. If a CLI update is available, the flexmonster version command notifies about it.

Options

The flexmonster version command can be run with the following options:

Option	Description
--help, -h	Shows the short guidance on this command in the console.

What’s next?

You may be interested in the following articles:

- flexmonster create (https://www.flexmonster.com/doc/flexmonster-create/)
- flexmonster add (https://www.flexmonster.com/doc/flexmonster-add/)
- flexmonster update (https://www.flexmonster.com/doc/flexmonster-update/)
- flexmonster help (https://www.flexmonster.com/doc/flexmonster-help/)

10.7. flexmonster help

flexmonster help|h [options]

The Flexmonster CLI command that provides help on all the available commands and options.

Options

The flexmonster help command can be run with the following options:

Option	Description
--help, -h	Shows the short guidance on this command in the console.

What’s next?

You may be interested in the following articles:

- flexmonster create (https://www.flexmonster.com/doc/flexmonster-create/)
- flexmonster add (https://www.flexmonster.com/doc/flexmonster-add/)
- flexmonster update (https://www.flexmonster.com/doc/flexmonster-update/)
- flexmonster version (https://www.flexmonster.com/doc/flexmonster-version/)

11. Documentation for older versions

The documentation from version 2.7 is available at the following links:

- Documentation 2.7 (https://s3.amazonaws.com/flexmonster/docs/Flexmonster-Documentation-2.7.pdf)
- API Reference 2.7 (https://s3.amazonaws.com/flexmonster/docs/Flexmonster-API-Reference-2.7.pdf)

The documentation from versions 2.4, 2.5, and 2.6 is available at the following links:

- Documentation 2.4, 2.5, and 2.6 (https://s3.amazonaws.com/flexmonster/docs/Flexmonster-Documentation-2.6.pdf)
- API Reference 2.4, 2.5, and 2.6 (https://s3.amazonaws.com/flexmonster/docs/Flexmonster-API-Reference-2.6.pdf)

The documentation from version 2.3 is available at the following links:

- (https://s3.amazonaws.com/flexmonster/docs/Flexmonster-Documentation-2.3.pdf)Documentation 2.3 (https://s3.amazonaws.com/flexmonster/2.3/docs/Flexmonster-Documentation-2.3.pdf) (https://s3.amazonaws.com/flexmonster/docs/Flexmonster-Documentation-2.3.pdf)
- (https://s3.amazonaws.com/flexmonster/docs/Flexmonster-API-Reference-2.3.pdf)API Reference 2.3

(https://s3.amazonaws.com/flexmonster/2.3/docs/Flexmonster-API-Reference-2.3.pdf)
(https://s3.amazonaws.com/flexmonster/docs/Flexmonster-API-Reference-2.3.pdf)

The documentation from version 2.2 is available at the following links:

- Documentation 2.2 (https://s3.amazonaws.com/flexmonster/2.2/docs/Flexmonster-Documentation-2.2.pdf)
- (https://s3.amazonaws.com/flexmonster/docs/Flexmonster-API-Reference-2.2.pdf)API Reference 2.2 (https://s3.amazonaws.com/flexmonster/2.2/docs/Flexmonster-API-Reference-2.2.pdf) (https://s3.amazonaws.com/flexmonster/docs/Flexmonster-API-Reference-2.2.pdf)

The Flex API reference is available at the following link:

- (/flex-api/)Flex API Reference (https://www.flexmonster.com/flex-api/) (/flex-api/)