Pivot Table & Charts Component Documentation

by Flexmonster

1. Getting started

Welcome to JavaScript Pivot Table & Charts component — web client-side component designed to view, analyze and manage multidimensional data online. The data is represented in compact yet interactive visual reports — multidimensional tables and charts that are fully customizable for client's needs.

The Component has 2 main advantages over competing types of applications:

- Usability and speed of the desktop application,
- Mobility and scalability of the web-based application.

Pivot Table enables you to analyze numerical data. With Pivot Table, you can look at the same information in different ways with just a few mouse clicks.

Documentation guide

- Look at the structure(#component)
  of our Component.
- Read our guide for embedding(/doc/how-to-create-js-pivottable/)
  Flexmonster Component into the web application.
- Find detailed tutorials to configure Pivot Table with your data source: JSON(/doc/json-data-source/)
  , CSV(/doc/csv-data-source/)
  , SQL database(/doc/connecting-to-relational-database/)
  , Microsoft Analysis Services(/doc/connecting-to-microsoft-analysis-services/)
  , Mondrian(/doc/connecting-to-pentaho-mondrian/)
  and icCube(/doc/connecting-to-iccube/)
  .
- Explore what is a report(/doc/configuring-report/)
  and how to configure it.
- Learn about integration with popular frameworks and technologies: Angular(/doc/integration-with-angularjs/)
  , Angular 2(/doc/integration-with-angular-2/)
  , React(/doc/integration-with-react/)
  , RequireJS(/doc/integration-with-requirejs/)
  , TypeScript(/doc/integration-with-typescript/)
  , ASP.NET(/doc/integration-with-asp-net/)
  and JSP(/doc/integration-with-jsp/)
  .
- Follow our guides(/doc/integration-with-charts/)
  about connecting with popular charting libraries.
- Customize Toolbar(/doc/customizing-toolbar/)
  and CSS styles(/doc/customizing-appearance/)
  of Flexmonster Component.
- Set localization(/doc/localizing-component/)
  for Pivot Table, read about different ways of export(/doc/export-and-print/)
  and look through our extensive API(/api/)
  .

Structure of Flexmonster Component

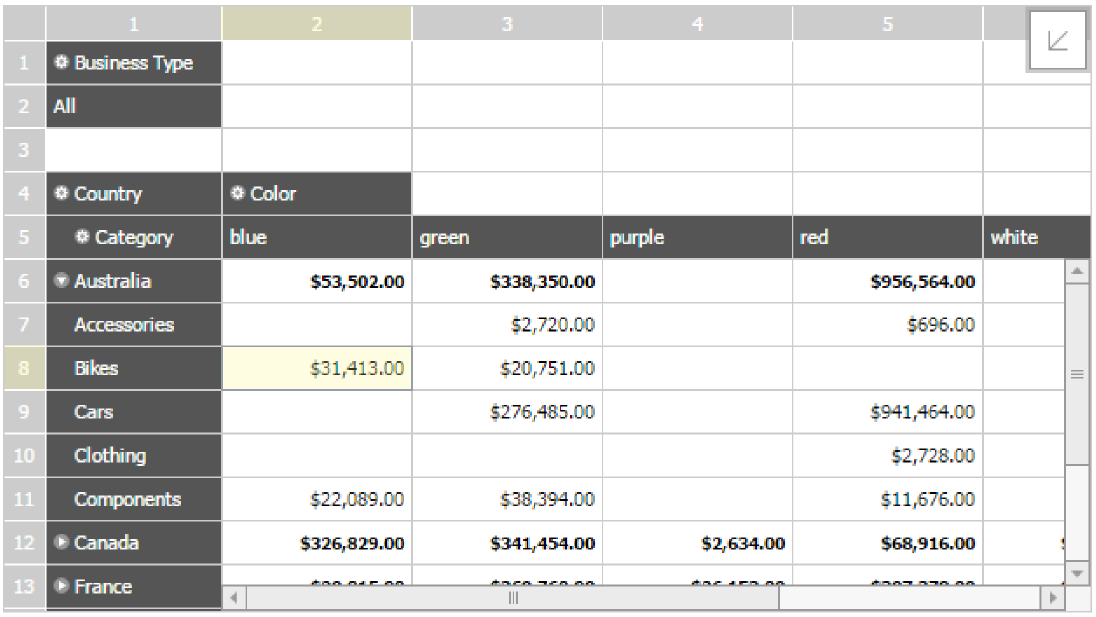
Our Component consists of the following functional parts:

- Pivot Grid(#grid)

- Pivot Chart(#chart)
- Fields List(#fieldslist)
- Filter(#filterwindow)
- Toolbar(#toolbar)

Pivot Grid

Pivot Grid shows the subset of data that can be defined in the report or configured by the end user. Apply sorting, filtering and other operations to make data look the way you need.

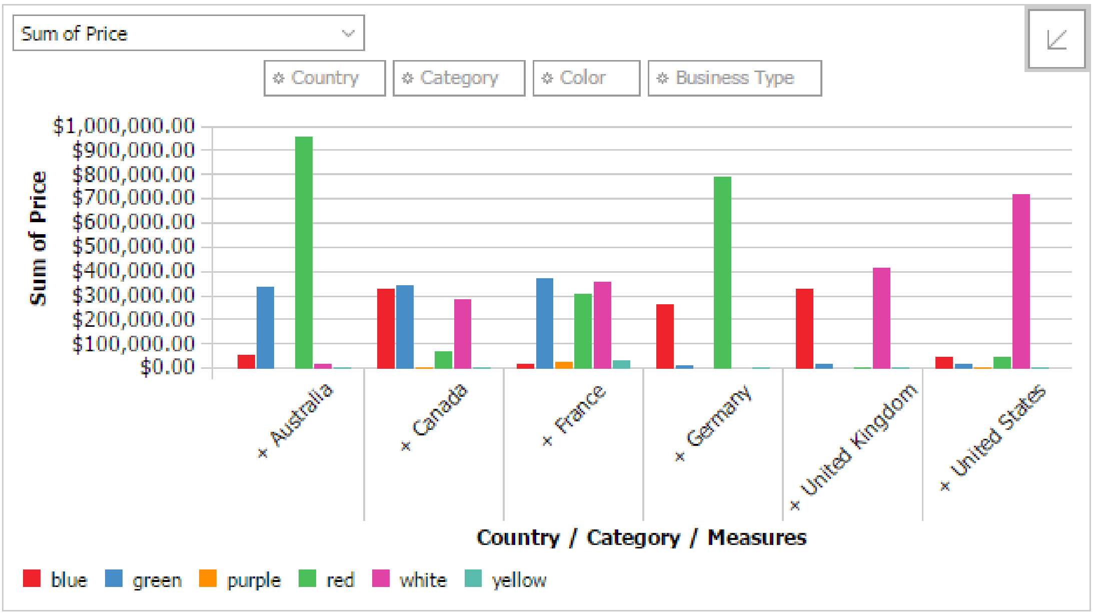


	1	2	3	4	5	
1	Business Type					
2	All					
3						
4	Country	Color				
5	Category	blue	green	purple	red	white
6	Australia	$53,502.00	$338,350.00		$956,564.00	
7	Accessories		$2,720.00		$696.00	
8	Bikes	$31,413.00	$20,751.00			
9	Cars		$276,485.00		$941,464.00	
10	Clothing				$2,728.00	
11	Components	$22,089.00	$38,394.00		$11,676.00	
12	Canada	$326,829.00	$341,454.00	$2,634.00	$68,916.00	
13	France					

Pay attention to the arrow in the upper right corner. This is the button used to show or hide Fields List view.

Pivot Chart

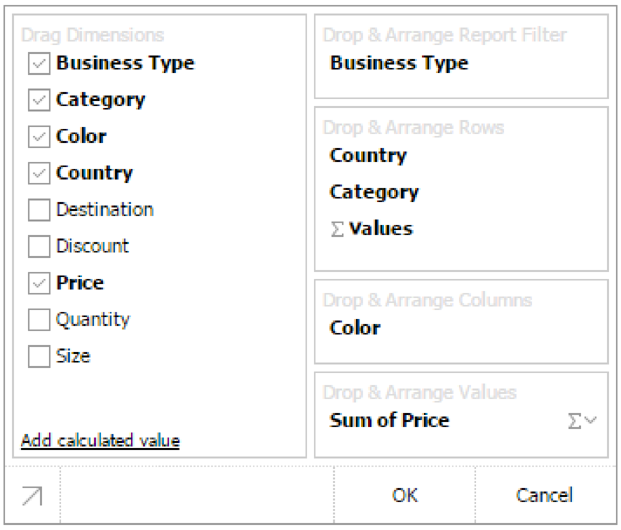
Pivot Chart displays your data in a convenient visual form. Our charts are interactive: expand the values by clicking on X axis and legend elements or see underlying data via drill through pop-up. The component supports the following chart types: Bar, Horizontal Bar, Line, Scatter, Pie, Bar Line, Bar Stack.



Fields List and Filters for charts are available as well as for the grid.

Fields List

Use **Fields List** to configure any view with drag-and-drop.

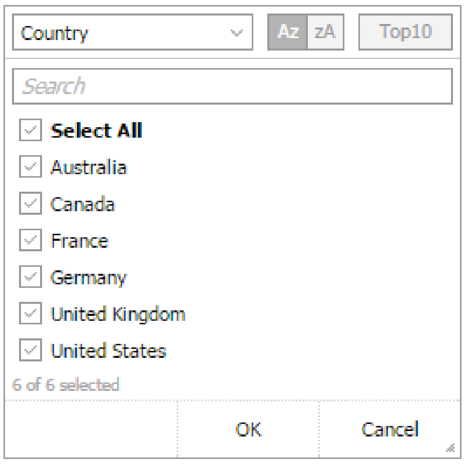


This window contains five main sections:

- All **dimensions** from the original data source;
- Fields selected or set inside the report to **Report filters** define which subset of data will be shown in a report;
- Fields selected or set inside the report to **Rows** define the left headers of the table;
- Fields selected or set inside the report to **Columns** define the upper headers of the table;
- Fields selected or set inside the report to **Values** define which measures will be shown on the grid. You need to choose aggregation function for each measure.

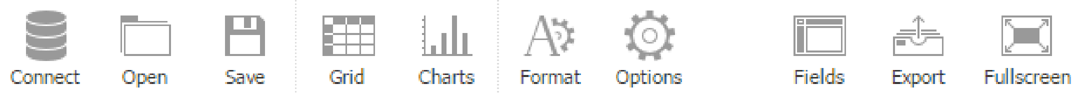
Filter

Sort and filter by members' names using **Filter**. Try filtering by value to get top X or bottom X results. Filter is accessible through the opening columns/rows filter controls and page filter controls on the grid and on charts.



Toolbar

Toolbar uses API calls and provides easy access to the most used features. The standard version of Toolbar looks the following way:



In addition, Toolbar can be customized. You can remove tabs and buttons or add new ones.

1.1. Quickstart

Embed into the web application

Here you can find a guide through the process of embedding Pivot Table and Charts Component into your HTML using flexmonster.js library.

To get your Pivot Table component embedded you should accomplish the following steps:

1. Download(/download-page/)
   a free trial of the component and extract the files from the downloaded package.
2. Copy flexmonster/ folder into your web project root to your server.
3. Include jquery and flexmonster.js into your HTML page. Please note, that your page should be in the same folder as flexmonster/:

```
<script src="flexmonster/lib/jquery.min.js"></script>
<script src="flexmonster/flexmonster.js"></script>
```

4. Create <div> container for the component:

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

```
<script src="flexmonster/lib/jquery.min.js"></script>
<script src="flexmonster/flexmonster.js"></script>
```

5. Add simple script to embed the component. Please note, licenseKey is your license or trial key, so you should replace XXXX-XXXX-XXXX-XXXX-XXXX with an actual key:

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

6. Now launch the page from browser — here you go! The component without data is embedded into your project. You can see the example on JSFiddle(http://jsfiddle.net/flexmonster/L54jrsp5/)
   .
7. The next step is to see your own data on the grid. Find detailed tutorials for connecting Pivot Table to the following data sources: JSON(/doc/json-data-source/)
   , CSV(/doc/csv-data-source/)
   , SQL database(/doc/connecting-to-relational-database/)
   , Microsoft Analysis Services(/doc/connecting-to-microsoft-analysis-services/)
   , Mondrian(/doc/connecting-to-pentaho-mondrian/)
   and icCube(/doc/connecting-to-iccube/)
   .

Should you have any problems — visit our troubleshooting page(/doc/troubleshooting/)
.

Initial jQuery call to embed the component

This is the first API call you need to know.

```
$("#pivotContainer").flexmonster({
 componentFolder:String,
 global:Object,
 width:Number,
 height:Number,
 report:Object|String,
 toolbar:Boolean,
 licenseKey:the license key})
```

As a parameter jQuery call gets id of the HTML element you would like to have as a container for the component. It embeds the component into HTML page and allows you to provide it with all necessary information for the initialization by setting the following arguments:

- componentFolder – URL of the component’s folder which contains all necessary files. Also, it is used as a

base URL for report files, localization files, styles and images. The default value for componentFolder is flexmonster/.
- global – object that allows you to preset options for all reports. These options can be overwritten for concrete reports. Object structure is the same as for report.
- width – width of the component on the page (pixels or percent). The default value for width is 100%.
- height – height of the component on the page (pixels or percent). The default value for height is 500.
- report – property to set a report. It can be inline report JSON or URL to report JSON. XML reports are also supported in terms of backward compatibility.
- toolbar – parameter to embed the toolbar (the one that is shown in Pivot Table demo(/demos/pivot-table-js/)
) or not. Default value is false – without toolbar.
- licenseKey – the license key.

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

1.2. System requirements

- Web Browser. Minimum browser requirements are listed below. Whichever browser you prefer, it is recommended that you use the most up-to-date version available for the best experience.
    - Chrome 12+
    - Firefox 15+
    - Internet Explorer 10+
    - Opera 15+
    - Safari 6.1+
    - iOS Safari 5.1.1+
- JavaScript must be enabled.
- Minimal recommended size for Pivot Component is 400×300px.
- jQuery 1.7+
- jQuery UI 1.9.2+

1.3. Managing license keys

All Flexmonster Pivot Table & Charts Component editions need a license key.

You can set a license key for the component via licenseKey parameter in JavaScript.

When you download a trial package, you receive a trial key within the package.

It is in index.html – set inline via licenseKey.
When you purchase the license, the trial key can be replaced by the license key.

Types of keys

There are the following types of keys:

- **Trial key** is provided within the trial package to try the component. The component runs with a trial key **without any restrictions**, with all capabilities according to the downloaded edition. The only limitation is that the integration with the 3rd party charting libraries is not available with the trial key. This key **is temporary** and has an expiration date. Usually, trial period lasts for 30 days.
- **Development license key** is provided for development purposes. It is applicable to your local host environment. Using development key, the component can be run locally on your computer **(localhost)** or

on the server using **IP address**. We do not license on per-developer basis, thus, you need only one license for the whole development team that works for the same project. This key is issued right after the purchase. Development license key can be limited in time or perpetual – depending on the purchased license.
- **Production license key** is provided for the production environment (domain/URL) where the component runs (e.g. yourcompany.com). It is not obligatory to know the production domain at the moment of purchase. It is needed when you publish your application on the web, so you can ask for the production key later. This key is **tied up to your domain name**. Production license key can be limited in time or perpetual – depending on the purchased license.
- **Staging key** can be issued by request after Flexmonster license purchase for testing on the real domain, but not in production environment.

You will receive development key right after Flexmonster license purchase. Also, as soon as you know the target domain/URL, the production key will be issued.

Component's editions

Flexmonster Pivot Table & Charts Component is available in 5 editions, you can choose from: Pivot Table for MS Analysis Services, Pivot Table for Pentaho Mondrian, Pivot Table for SQL/CSV/JSON Unlimited, Pivot Table for SQL/CSV/JSON Basic and Enterprise Bundle.

Each edition has its own key which unlocks it. There are some differences between them. Please, see the details below:

- **Pivot Table for MS Analysis Services** works with data from Microsoft Analysis Services OLAP cubes.
- **Pivot Table for Pentaho Mondrian** supports data from Mondrian or icCube.
- **Pivot Table for SQL/CSV/JSON Unlimited** supports data from SQL Databases or static files. It has no restrictions for CSV, OCSV and JSON data size.
- **Pivot Table for SQL/CSV/JSON Basic** supports data from SQL Databases or static files. It has the following limits on data sizes: CSV data sources (up to 5MB), OCSV data sources (up to 2MB), JSON data sources (up to 5MB) and inline JSON (up to 100K records).
- **Enterprise Bundle** supports all data sources, including SQL Databases, CSV files, JSON, MS Analysis Services, Mondrian and icCube. This edition has no restrictions for data size.

For prices and further details for each component's edition please visit Pricing(/pricing)
page.

1.4. Updating to the latest version

Our customers have an opportunity to update Flexmonster Pivot Table & Charts component to the latest version. This is a very simple process requiring just a couple of steps:

- Go to our Download(/download-page)
  page and download a trial package of the corresponding version.
- Unzip the files from the package.
- Replace your flexmonster/ folder with the new one (find it in component/ directory or in client/ directory – depending on the trial package you downloaded).
- If you are updating from the previous major version to 2.3, please read our Migration guide(/doc/migration-guide)
  .
- To make sure 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.
- If you use our **Accelerator**, please find instructions in the separate section(/doc/updating-to-the-latest-

version/#accelerator)
below.
- If you use our **Compressor**, please find instructions in the separate section(/doc/updating-to-the-latest-version/#compressor)
below.

Now it's done. You are welcome to use the most recent version of Flexmonster Pivot Table & Charts with its newest features.

NOTE! Being our active customer (during 1 year after purchase or renewal payment) you are able to **update for free**. If being our client you would like to renew the maintenance, please contact us(/contact)
for more details.

Updating Flexmonster Accelerator

Microsoft Analysis Services / EXE version

- Stop Accelerator by closing the program.
- Replace flexmonster-proxy-ssas.exe with updated version from the package (find it in server/ directory).
- Start Accelerator by launching flexmonster-proxy-ssas.exe.

Microsoft Analysis Services / Windows Service version

- Reinstall Accelerator using updated MSI installer from the package (find it in server/installer/ directory).

Pentaho Mondrian

- Stop Accelerator by closing the program.
- Replace flexmonster-proxy-mondrian.jar with updated version from the package (find it in server/ directory).
- Start Accelerator by executing java -jar flexmonster-proxy-mondrian.jar in terminal.

Updating Flexmonster Compressor

.NET

- Replace Flexmonster.Compressor.dll with updated version from the package (find it in server/.net/ directory).

Java

- Replace flexmonster-compressor.jar with updated version from the package (find it in server/java/ directory).

PHP

- Replace flexmonster-compressor.php with updated version from the package (find it in server/php/ directory).

1.5. Typical errors

License keys

Current key is only applicable for example.com. You are trying to use the following key: XXXX-XXXX-XXXX-XXXX-XXXX

Please verify that the domain name shown in your error message (e.g. example.com) matches the domain name of your project for which you requested the key. In case they are different, you should contact our client service team(/contact/)
.

Integration with third-party charting libraries is not available in the trial version.

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

Your license key is outdated and will not function with the current version. Please, contact support team to find out about upgrade options.

In case you are updating from the previous major version to 2.3, the new license keys are required. If your maintenance is active – please request new license keys for free(/contact/)
.

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

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

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

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

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

This message means your trial license key has expired. To carry on the evaluation of our component please download a new trial package. For other special options, you can contact our sales team(/contact/)
.

Invalid license key! You are trying to use the following key: XXXX-XXXX-XXXX-XXXX-XXXX

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

Serial number is corrupted!

This message means that you probably have copied only a part of the key. Please open the message you received after the purchase and copy the license key once more.

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 resource requested.

This message may refer to one of the reasons below:

- There is no file with such name on the server. Please make sure the filename is correct.
- The browser requires more privileges to load the data. By default, cross-domain requests from JavaScript are blocked. Please enable CORS to make such requests. You may refer to enable-cors.org(https://enable-cors.org/)
  to resolve the issue.
- Internal server error — please open browser’s console and check the errors here.

CSV, OCSV and JSON data sources aren't supported in the current edition.

Please check that the license key data source type, the type of the downloaded package and the data source type you actually use correspond to each other. In case you want to test some other data source, which is not

included in your licensing plan, leave a request via Contact Us form(/contact/)
.

MS OLAP support is not available in the current edition.

Please check that the license key data source type, the type of the downloaded package and the data source type you actually use correspond to each other. In case you want to test some other data source, which is not included in your licensing plan, leave a request via Contact Us form(/contact/)
.

Mondrian support is not available in the current edition.

Please check that the license key data source type, the type of the downloaded package and the data source type you actually use correspond to each other. In case you want to test some other data source, which is not included in your licensing plan, leave a request via Contact Us form(/contact/)
.

icCube support is not available in the current edition.

Please check that the license key data source type, the type of the downloaded package and the data source type you actually use correspond to each other. In case you want to test some other data source, which is not included in your licensing plan, leave a request via Contact Us form(/contact/)
.

File is too large!

This message means that your edition is Pivot Table for SQL/CSV/JSON Basic. Such edition has 5 MB limitation on data size. For uploading bigger files you should upgrade to the Unlimited edition. For further details please contact our client service team(/contact/)
.

Error opening URL. Please check Internet conne?tion.

This error means one of the following:

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

Invalid datasource or catalog. Please check.

Check catalog and cube names you entered for connection to the cube once more. To get the exact names please use our OLAP connection tool from the toolbar. Click 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 the Data Source Info and 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. Please try opening the browser's console and checking here.

Stream error occurred while loading example.com

This error may be caused by one of the reasons below. Please check all of them:

- Check once more that your filename/path to proxy URL to the OLAP data source/catalog and cube names exist and the current user has enough rights to access them.
- Make sure that cross-domain requests are allowed. Find additional information on this resource: 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 MSAS via XMLA protocol, please 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/)
  . Also, we suggest trying our special server-side proxy Accelerator instead of XMLA (read more(/doc/getting-started-with-accelerator-ssas/)
).

1.6. Migration guide

From Flexmonster Pivot Table & Charts 2.2 to 2.3

The new version of Pivot Table & Charts offers a couple of new methods. Some existing methods were improved and some were removed. This guide will help you to migrate from the previous version to the new one.

To migrate from the previous major version to 2.3 you should read the following:

1. License keys(/doc/migration-guide/#keys)
2. How to embed pivot component into your website(/doc/migration-guide/#embedPivotComponent)
3. API updates(/doc/migration-guide/#updates)
4. New methods(/doc/migration-guide/#added)
5. Removed methods(/doc/migration-guide/#removed)
6. New structure of report object(/doc/migration-guide/#structure)
7. Event list(/doc/migration-guide/#eventList)
8. Toolbar localization(/doc/migration-guide/#toolbarLocalization)
9. Folder structure(/doc/migration-guide/#folderStructure)
10. Documentation for version 2.2(/doc/migration-guide/#docs)

License keys

Version 2.3 requires new license keys. If you have active maintenance – contact our support team and you will get 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 pivot component 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 the following way:

```
<script src="flexmonster/lib/jquery.min.js"></script>
<script src="flexmonster/flexmonster.js"></script>
```

In the previous version embedPivotComponent() function was used to embed the component. In the version 2.3 it was removed, but it will be supported in this version in terms of backward compatibility. Instead of embedPivotComponent() you should use a jQuery call:

```
var pivot = $("#pivotContainer").flexmonster({
    componentFolder:String,
    global:Object,
    width:Number,
    height:Number,
    report:Object|String,
    toolbar:Boolean,
    licenseKey:String})
```

As a parameter jQuery call gets a selector of the HTML element you would like to have as a container for the component.

All the properties are optional in a configuration object. If you run $("#pivotContainer").flexmonster() – an empty component without toolbar will be added with the default width and height. The configuration object passed to jQuery call can have the following properties:

- componentFolder – URL of the component's folder which contains all necessary files. Also, it is used as a base URL for report files, localization files, styles and images. The default value for componentFolder is flexmonster/.
- global – object that allows you to preset options for all reports. These options can be overwritten for concrete reports. The structure of the object is the same as for report.
- width – width of the component on the page (pixels or percent). The default value for width is 100%.
- height – height of the component on the page (pixels or percent). The default value for height is 500.
- report – property to set a report. It can be inline report JSON or URL to report JSON. XML reports are also supported in terms of backward compatibility. In the previous version this property was called configUrl. The structure of report object was changed. Chapter New structure of report object(/doc/migration-guide/#structure)
  explains how to use the new structure.
- toolbar – parameter to embed the toolbar (the one that is shown in Pivot Table demo(/demos)
) or not. The default value is false – without toolbar (in the previous version this property was called withToolbar).
- licenseKey – the license key.

Event handlers can also be set as properties for the jQuery call. Check the list here(/doc/migration-guide/#eventList)
.

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 of the component by selector as following: 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 XML format. Now save returns report as JSON object. See New structure of report object(/doc/migration-guide/#structure)
   for further details.
2. getAllMeasures and getMeasures
   calculated property was removed from the measure object. To see, if some measure was calculated, check 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, you need to change its name everywhere 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 you should change

   ```
   options.showTotals = false;
   ```

to

```
options.grid.showTotals = false;
```

You need to do the same for all the renamed options. All references to removed options should be deleted. You can find the renamed options among other changes of report object here(/doc/migration-guide/#structure)
.

4. setHandler
   setHandler was replaced with on() and off(). Event list was extended. Check the full list here(/doc/migration-guide/#eventList)
   .
5. getReport and setReport
   Object structure was changed. Chapter New structure of report object(/doc/migration-guide/#structure) explains how to use the new structure.

New methods

These methods were added to the version 2.3:

1. dispose() – prepares the pivot table instance to be deleted with the browser's garbage collection.
2. getData(options: Object, callbackHandler: Function, updateHandler:Function) – enables the integration with the 3rd party charting libraries (FusionCharts, Highcharts, Google Charts, other). Returns the data from the pivot table component. options object can have slice property. If slice in options is not defined, the API call will return data displayed in the pivot table.
3. off(eventName:String, functionName:String) – removes event listener. If functionName property is not defined, all event listeners will be removed.
4. updateData(params:Object|Array) – addJSON and connectTo API calls were replaced by this API call. It is used to update data for the report without cleaning the report.

More detailed information about each API call you can find in API Reference(/api/)
.

Removed methods

1. addMeasure should be removed. It can be replaced with addCalculatedMeasure(measure)(/api/addcalculatedmeasure/)
   .
2. addDataArray, addDataRecord, addDimension and addHierarchy should be removed. JSON data source can be used instead. The first element of JSON is used to define data types, captions, group fields under one dimension, define 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()(/api/getcell/)
   .

   ```
   flexmonster.getCell(1,1);
   ```

9. gridColumnCount
10. gridRowCount
11. percentZoom
12. removeAllMeasures should be removed. It can be replaced with removeAllCalculatedMeasures()(/api/removeallcalculatedmeasures/)
    .
13. removeMeasure should be removed. It can be replaced with removeCalculatedMeasure(name)(/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 two ways:
    - using global.localization inside 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

New structure of report object

In version 2.3 report object was structured. Properties were logically grouped into sub-objects. For example, all related to data sources properties are inside dataSource sub-object now.

JSON report objects of the previous version and XML reports are fully supported in terms of backward compatibility. It means you can open or set reports from the previous major version. Please note, that all reports will be saved in new JSON structure. Thus, the component can be used to migrate to the new version of reports. Also, we created a small utility(/blog/moving-to-json-converter-for-old-xml-reports/)
for converting XML reports to JSON. That is an easy and convenient way of migrating to the new format.

Here is an example of 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 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
recordsetDelimiter	dataSource.recordsetDelimiter
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

Please note:

- showFiltersExcel was removed from options. Instead, you can use showFilter parameter of export options inside 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 report object
- expandAll was removed from options and moved to slice inside report object
- ignoreQuotedLineBreaks was removed from options and moved to dataSource inside 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 OLAP structure or data from the report, localization file and all data source files were loaded successfully and the grid/chart was 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 Fields List is opened.
- jsFieldsListCloseHandler was renamed to fieldslistclose – String, triggered when the built-in Fields 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 loaded 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 the 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 Open file dialog was opened and customer clicks Cancel button. It happens when:
    1. customer uses menu Connect -> to local CSV/JSON
    2. API method updateData()(/api/updateData/)
       is called with parameter browseForFile = true
- dataloaded – String, triggered when the component loaded data. Please use olapstructureloaded for OLAP data sources.
- loadingdata – String, triggered when data starts loading from local or remote CSV, JSON. Please use loadingolapstructure for OLAP data sources.
- loadinglocalization – String, triggered when localization file starts loading.
- loadingolapstructure – String, triggered when the structure of OLAP cube starts loading.

- loadingreportfile – String, triggered when a report file started loading.
- localizationerror – String, triggered when some error appeared while loading localization file.
- localizationloaded – String, triggered when localization file was loaded.
- olapstructureerror – String, triggered when some error appeared while loading OLAP structure.
- olapstructureloaded – String, triggered when OLAP structure was loaded.
- openingreportfile – String, triggered when:
    1. customer uses menu Open -> Local report
    2. API method open()(/api/open/)
       is called
- querycomplete – String, triggered after the data query was complete.
- queryerror – String, triggered if some error occurred while running the query.
- ready – String, triggered when the component's initial configuration completed and the component is ready to receive API calls.
- reportfilecancelled – String, triggered when the customer uses menu Open -> Local report and clicks Cancel button.
- reportfileerror – String, triggered when some error occurred during the loading of the report file.
- runningquery – String, triggered before 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 it is necessary to load data from the external data storage in case of OLAP data source. Data query is started when:
    1. Slice was changed;
    2. Any filter was used;
    3. Drill up/drill down was used;
    4. Columns or rows were expanded;
    5. Drill through was used.

Toolbar localization

In the previous version, toolbar could be localized by passing translations to toolbarLocalization parameter in embedPivotComponent(). Now the localization of toolbar is set inside localization file. To set localization only for 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 Pivot Table, please read Localizing component(/doc/localizing-component)
.

Folder structure

FOLDERS	CONTENTS
assets/	Contains the theme images
lib/	Includes additional JavaScript libraries
toolbar/	Contains toolbar files: images, JS, and CSS
flexmonster.css	The theme CSS
flexmonster.js	The main Pivot Table JavaScript file

Please note, that folder structure was changed. Now html5-assets/ is replaced with assets/, lib/ and flexmonster.css.

Documentation for version 2.2

In case 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)
.

1.7. Installation troubleshooting

If you are facing any problems with the embedding of the component, in your browser go to the page where the Flexmonster component should be displayed and open the browser's console to check if you have any errors in the console. Try the following steps:

- If you get the alert "Flexmonster: jQuery is not loaded.", check the path to jquery.min.js 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.js 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 componentFolder parameter to your $.flexmonster() call. Pay attention to the message "Files not found: ..." and set componentFolder to receive the correct paths to lib/jqueryui.min.js, lib/jqueryhelpers.min.js and flexmonster.css.

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

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

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

2. JSON data source

This article illustrates how to build a report based on JSON data source. JSON data source is good to display data which already is on your page. This is the most simple case of the configuration. Also, you can load JSON data from a file (please see Connect > To local JSON in the toolbar) or refer JSON file as data source.

The component supports a certain format of JSON – array of objects, where each object is an unordered set of name/value pairs. For example:

```
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
    }
];
```

You need to accomplish the following steps to use JSON data as a data source in pivot table embedded into your project.

1. You already have in your HTML page an empty component. If Flexmonster Component is not embedded – return to Quickstart(/doc/how-to-create-js-pivottable/)
   . Your code should look like the following example:

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

   <script>
    var pivot = $("#pivotContainer").flexmonster({
     toolbar: true,
     licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
    });
   </script>
   ```

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

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

   <script>

    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 = $("#pivotContainer").flexmonster({
   toolbar: true,
   licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
  });
 </script>
```

3. Add data property to report object:

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

<script>
 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 = $("#pivotContainer").flexmonster({
  toolbar: true,
  report: {
   dataSource: {
    data: jsonData
   }
  },
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

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

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

```
<script>
 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 = $("#pivotContainer").flexmonster({
  toolbar: true,
  report: {
   dataSource: {
    data: jsonData
   },
   slice: {
    rows: [
    { uniqueName: "Color" },
    { uniqueName: "[Measures]" }
    ],
    columns: [
    { uniqueName: "Country" }
    ],
    measures: [
    {
     uniqueName: "Price",
     aggregation: "sum"
    }
    ]
   }
  },
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

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

Now you know how to build a report based on our sample JSON. The next step for you is to visualize your JSON data.

How to display non-English characters correctly

If you use the 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 JSON files also must be encoded as UTF-8.
2. In case you are not able to change the content of your HTML file, you can embed Flexmonster Component in a separate JS file with specified UTF-8 encoding.

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

2.1. Data types in JSON

In addition, the first object in JSON array can be used to define data types, captions, group fields under one dimension, define hierarchy with 3 levels, etc. Here is the list of supported properties that can be used in the first object of input array:

- type – data type. Can be:
  - "string" – field contains string data. You will be able to aggregate it only with Count and Distinct Count aggregations. It will be sorted as string data.
  - "number" – field contains numeric data. You will be able to aggregate it with all different aggregations. It will be sorted as numeric data.
  - "level" – field is a level of hierarchy. This type is used together with other properties such as: hierarchy, level and parent
  - "month" – field contains months.
  - "weekday" – field contains days of the week.
  - "date" – field is a date. Such field will be split into 3 different fields: Year, Month, Day.
  - "date string" – field is a date. Such field will be formatted using date pattern (default is dd/MM/yyyy).
  - "year/month/day" – field is a date. You will see such date as a hierarchy: Year > Month > Day.
  - "year/quarter/month/day" – field is a date. You will see such date as a hierarchy: Year > Quarter > Month > Day.
  - "time" – field is a time (numeric data). Such field will be formatted using HH:mm pattern. Min, Max, Count and Distinct Count aggregations can be applied to it.
  - "datetime" – field is a date (numeric data). Such field will be formatted using dd/MM/yyyy HH:mm:ss pattern. Min, Max, Count and Distinct Count aggregations can be applied to it.
  - "id" – field is an id of the fact. Such field is used for editing data. This field will not be shown in Fields List.
  - "hidden" – field is hidden. This field will not be shown in Fields List.
- caption – hierarchy caption.
- hierarchy – hierarchy name, if the field is a level of hierarchy ("type":"level").
- level – caption of the level, if the field is a level of hierarchy ("type":"level").
- parent – caption of the parent level, if the field is a level of hierarchy ("type":"level").
- dimensionUniqueName – dimension unique name. Can be used to group several fields under one dimension.
- dimensionCaption – dimension caption. Is used as a display name (folder name in Fields List) when several fields are grouped under one dimension.

For example, you can add the following first object in JSON array and see how it changes the report:

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

<script>
    var jsonData = [
        {
            "Color": {type: "string"},
   "Country":{type: "level", hierarchy: "Geography",
     level: "Country"},
   "State": {type: "level", hierarchy: "Geography",
     level: "State", parent: "Country"},
   "City": {type: "level", 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 = $("#pivotContainer").flexmonster({
  toolbar: true,
  report: {
   dataSource: {
    data: jsonData
   },
   slice: {
    rows: [
    { uniqueName: "Color" },
    { uniqueName: "[Measures]" }
    ],
    columns: [
    { uniqueName: "Geography" }
    ],
    measures: [
    { uniqueName: "Price", aggregation: "sum" }
    ]
   }
  },
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Try on JSFiddle(http://jsfiddle.net/flexmonster/mr0Lwkab/)
.

Undefined types in JSON

It is possible to define only necessary types of fields and leave all others empty {}. Type of these fields will be selected automatically. In the following example "Color" was left undefined:

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

<script>
    var jsonData = [
        {
            "Color": {},
            "Price": {type: "number"}
        },
        {
            "Color" : "green",
            "Price" : 174
        },
        {
            "Color" : "red",
            "Price" : 166
        }
    ];

    var pivot = $("#pivotContainer").flexmonster({
        toolbar: true,
        report: {
            dataSource: {
                data: jsonData
            }
        },
        licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
    });
</script>
```

Please note, that in Fields List will be available only two fields: "Color" and "Price". All other fields ("Country", "City" etc.) will be omitted because they were not mentioned in the first object of JSON array.

Supported date formats

To make date fields be interpreted as date, you should define data type, for example, "type":"date", "type":"date string", "type":"year/month/day" or "type":"year/quarter/month/day". Besides, data from these fields should have special date format to be understood properly.
Pivot table component supports ISO 8601(http://www.w3.org/TR/NOTE-datetime)
date (other formats may be used, but results can be unexpected). For example, "2016-03-20" (just date) or "2016-03-20T14:48:00" (date and time).

3. CSV data source

This article illustrates how to build a report based on CSV data source. CSV data source is good to display data which is exported from other sources (for instance, from Excel). Also, you can load CSV data from a file (please see Connect > To local CSV in the toolbar) or refer CSV file as data source.

You need to accomplish the following steps to use CSV data as a data source in pivot table embedded into your project.

1. You already have in your HTML page an empty component. If Flexmonster Component is not embedded – return to Quickstart(/doc/how-to-create-js-pivottable/)
   . Your code should look like the following example:

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

   <script>
    var pivot = $("#pivotContainer").flexmonster({
     toolbar: true,
     licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
    });
   </script>
   ```

2. Let’s say you have the following CSV file called data.csv. To make it simple, 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 filename property to report object:

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

   <script>
    var pivot = $("#pivotContainer").flexmonster({
     toolbar: true,
     report: {
      dataSource: {
       filename: "data.csv"
      }
     },
   ```

```
   licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
  });
 </script>
```

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

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

   <script>
    var pivot = $("#pivotContainer").flexmonster({
     toolbar: true,
     report: {
      dataSource: {
       filename: "data.csv"
      },
      slice: {
       rows: [
       { uniqueName: "Color" },
       { uniqueName: "[Measures]" }
       ],
       columns: [
       { uniqueName: "Country" }
       ],
       measures: [
       { uniqueName: "Price", aggregation: "sum" }
       ]
      }
     },
     licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
    });
   </script>
   ```

5. Now launch the page from browser — here you go! A pivot table based on CSV data is embedded into your project. Open this example on JSFiddle(http://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 the 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 CSV files also must be encoded as UTF-8.
2. In case you are not able to change the content of your HTML file, you can embed Flexmonster Component in a separate JS file with specified UTF-8 encoding.

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

3.1. Data types in CSV

Using CSV data source you can indicate how data will be interpreted by Pivot table component. For this, you can use special prefixes for columns names. Pivot table has the following prefixes for CSV files:

- + means that field is a dimension.
- - field is a value.
- m+ this prefix will indicate that the field is a month.
- w+ to indicate that the field is a day of the week.
- d+ field is a date. Such field will be split into 3 different fields: Year, Month, Day. Date formats that are supported by Pivot table is described below.
- D+ field is a date. You will see such date as a hierarchy: Year > Month > Day.
- D4+ field is a date. You will see such date as a hierarchy: Year > Quarter > Month > Day.
- ds+ field is a date. Such field will be formatted using date pattern (default is dd/MM/yyyy)
- t+ field is a time (measure). Such field will be formatted using HH:mm pattern
- dt+ field is a date (measure). Such field will be formatted using dd/MM/yyyy HH:mm:ss pattern
- id+ field is an id of the fact.

Here is the minimal CSV file which will treat Year as dimension, rather than numeric measure:

```
Country, +Year, Sales
US, 2010, 200
UK, 2010, 100
```

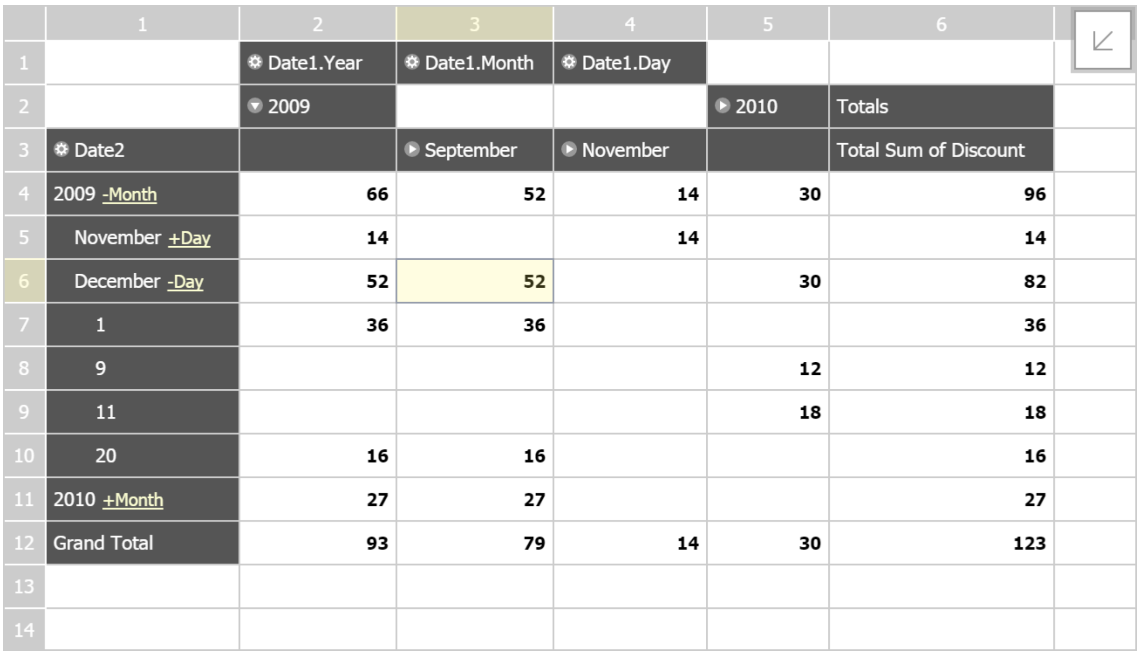
Supported date formats

To make date column be interpreted as date, you should use prefixes d+, D+ and D4+ for CSV columns. Besides, data from these columns should have special date format to be understood properly.
The component supports ISO 8601(http://www.w3.org/TR/NOTE-datetime)
date (other formats may be used, but results can be unexpected). For example, "2016-03-20" (just date) or "2016-03-20T14:48:00" (date and time).

Here is an example of the CSV file with date columns – Date1 and Date2:

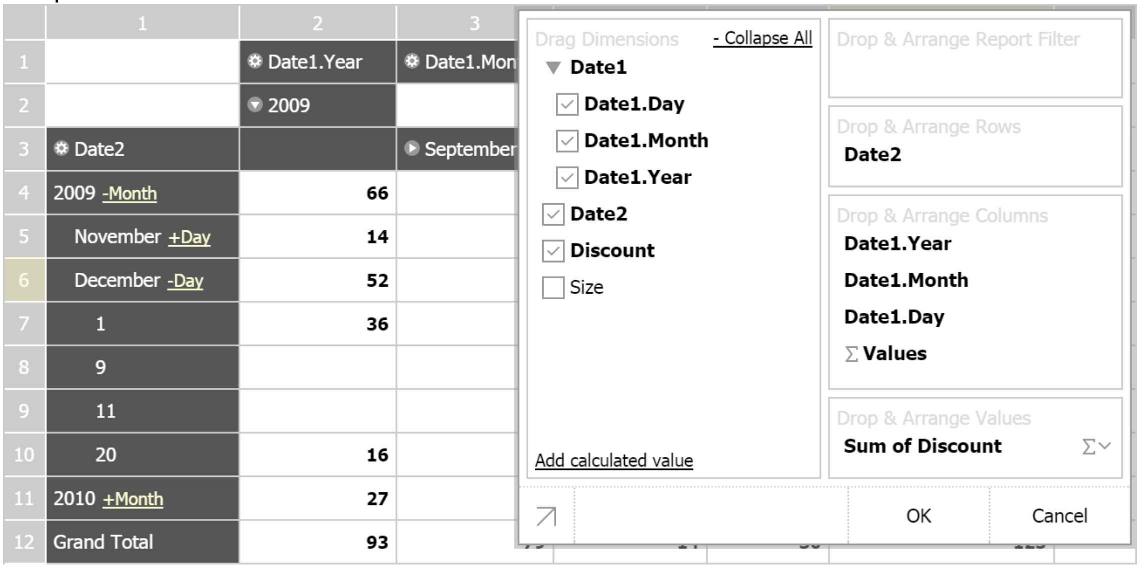
```
Size, Discount, d+Date1, D+Date2
214 oz, 14, 2009-11-01, 2009-11-09
214 oz, 12, 2010-12-09, 2009-12-09
212 oz, 36, 2009-09-01, 2009-12-01
212 oz, 27, 2009-09-01, 2010-12-02
212 oz, 18, 2010-11-09, 2009-12-11
212 oz, 16, 2009-09-01, 2009-12-20
```

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
13						
14						

As you can see, Date1 column with prefix d+ is split into three separate fields — Year, Month, Day. In Pivot component it will look as follows:



	1	2	3
1		Date1.Year	Date1.Mon
2		2009	
3	Date2		September
4	2009 -Month	66	
5	November +Day	14	
6	December -Day	52	
7	1	36	
8	9		
9	11		
10	20	16	
11	2010 +Month	27	
12	Grand Total	93	

Date2 column with D+ prefix is interpreted as a hierarchy that can be drilled down to months and days.

4. Connecting to SQL database

Everyone who has ever made reports or pivot tables with relational databases has most probably faced a problem with finding a suitable component that not only performs all required functions but also executes them fast. Flexmonster solved this problem and created **Flexmonster Data Compressor – a special server-side compression tool that helps you to increase data loading speed** from server to customer's browser.

Advantages of Flexmonster Compressor:

- Easy convert response from database with few lines of code

- Stream large datasets with minimum delay to the user
- Lightweight and simple format which does not require significant amount of RAM and CPU power
- Available for various platforms and databases

The alphabet with special characters will be displayed correctly because our Compressor sets encoding to UTF-8. Please make sure that your content-type HTTP header also have UTF-8 encoding:

```
header('Content-type: text/html; charset=utf-8');
```

Supported databases

- Oracle
- MySQL
- Microsoft SQL Server
- PostgreSQL
- Other ADO.NET / ODBC / JDBC databases

Guides

Flexmonster Compressor is available for .NET, Java and PHP. Follow detailed tutorials for each technology:

- Connecting to relational database with .NET(/doc/connecting-to-relational-database-with-net/)

- Connecting to relational database with .NET Core(/doc/connecting-to-relational-database-with-net-core/)

- Connecting to relational database with Java(/doc/connecting-to-relational-database-with-java/)

- Connecting to relational database with PHP(/doc/connecting-to-relational-database-with-php/)

4.1. Connecting to relational database with .NET

Requirements

- Flexmonster Pivot Component version 2.213 or higher
- Microsoft .NET Framework 4 or higher
- Driver for the database

Supported databases

- Oracle Database – driver(http://www.nuget.org/packages/Oracle.ManagedDataAccess/)

- MySQL – driver(http://www.nuget.org/packages/MySql.Data/)

- Microsoft SQL Server – driver is built-in
- PostgreSQL – driver(http://www.nuget.org/packages/Npgsql/)

- Other ADO.NET / ODBC databases

Step 1: Embed the component into your web page

You already have in your HTML page an empty component. If Flexmonster Component is not embedded – return to Quickstart(/doc/how-to-create-js-pivottable/)
. Your code should look like the following example:

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Step 2: Setup Data Compressor on the server

First of all, you need to add the following dependencies to your project:

- Appropriate database driver. Please note that Microsoft SQL Server driver is built-in.
- Flexmonster.Compressor.dll – it is located in the Download Package(/download-page)

Below is the connection and query sample for MS SQL Server:

```
string connectionString = "Server=localhost;Database=XXX;";
SqlConnection sqlConnection = new SqlConnection(connectionString);
sqlConnection.Open();
string query = "SELECT * FROM YYY";
DbCommand command = new SqlCommand(query, sqlConnection);
DbDataReader reader = command.ExecuteReader();
```

The following line of code will convert DbDataReader to compressed Stream:

```
Stream inputStream = Flexmonster.Compressor.CompressDb(reader);
```

Now, you can create a response from the Stream. For simple cases, it is enough to read all content:

```
string output = new StreamReader(inputStream).ReadToEnd();
```

Also, it is possible to create streaming response. It means that the end user will get the response almost without delay and server will use less amount of memory. It is recommended way for large datasets:

```
HttpResponseMessage response = Request.CreateResponse();
response.Content =
    new PushStreamContent((Stream outputStream, HttpContent content,
                           TransportContext context) =>
    {
        int length = 0;
        byte[] buffer = new byte[10240];
        while ((count = inputStream.Read(buffer, 0, buffer.Length)) > 0)
        {
            outputStream.Write(buffer, 0, length);
            outputStream.Flush();
        }
        outputStream.Close();
    }, new MediaTypeHeaderValue("text/plain")
);
```

Full project is available at the server/.net/ inside Download Package(/download-page)
.

Step 3: Enable cross-origin resource sharing (CORS).

By default, browser prevents JavaScript from making requests across domain boundaries. CORS allows web applications to make cross-domain requests. Here are some useful links explaining how to setup CORS on your server:

- IIS – CORS on IIS(http://enable-cors.org/server_iis7.html)

Step 4: Configure report with your own data

Now it’s time to configure Flexmonster Pivot Component on the web page. Let’s create a minimal report for this (replace filename and other parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "ocsv",
   /* URL to the Data Compressor .NET */
   filename: "http://localhost:55772/api/flexmonster/get"
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

Launch the web page from browser — here you go! A pivot table is embedded into your project. Check the example on JSFiddle(http://jsfiddle.net/flexmonster/y4jnrk1y/)
.

Examples

Saving to file

```
string connectionString = "Server=localhost;Database=XXX;";
SqlConnection sqlConnection = new SqlConnection(connectionString);
sqlConnection.Open();
string query = "SELECT * FROM YYY";
DbCommand command = new SqlCommand(query, sqlConnection);
DbDataReader reader = command.ExecuteReader();

Stream inputStream = Flexmonster.Compressor.CompressDb(reader);
FileStream fileStream = File.Create("data.ocsv");
inputStream.CopyTo(fileStream);
fileStream.Close();
```

4.2. Connecting to Relational Database with .NET Core

Requirements

- Flexmonster Pivot Component version 2.213 or higher
- Microsoft .NET Core 1.0 and 1.1
- Driver for the database

Supported databases

- Oracle Database
- MySQL
- Microsoft SQL Server – driver is built-in
- PostgreSQL
- Other ADO.NET / ODBC databases

Step 1: Embed the component into your web page

You already have in your HTML page an empty component. If Flexmonster Component is not embedded – return to Quickstart(/doc/how-to-create-js-pivottable/)
. Your code should look like the following example:

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Step 2: Setup Data Compressor on the server

First of all, you need to install Flexmonster Compressor. Please download it on nuget.org(https://www.nuget.org/packages/Flexmonster.Compressor/)
or run the following command in the Package Manager Console(https://docs.microsoft.com/en-us/nuget/tools/package-manager-console)
:

```
Install-Package Flexmonster.Compressor
```

Then add the appropriate database driver to your project. Below is the connection and query sample for MySql:

```
string connetionString = "server=localhost;database=XXX;";
string sql = "SELECT * FROM YYY";
MySqlConnection sqlConnection = new MySqlConnection(connetionString);
sqlConnection.Open();
MySqlCommand command = new MySqlCommand(sql, sqlConnection);
DbDataReader dataReader = command.ExecuteReader();
```

The following line of code will convert DbDataReader to compressed Stream:

```
Stream inputStream = Compressor.CompressDb(dataReader);
```

Now, you can create a response from the Stream. For simple cases, it is enough to read all content:

```
StreamReader reader = new StreamReader(inputStream);
string output = reader.ReadToEnd();
```

Step 3: Enable cross-origin resource sharing (CORS).

By default, browser prevents JavaScript from making requests across domain boundaries. CORS allows web applications to make cross-domain requests. Here are some useful links explaining how to setup CORS on your server:

- IIS – CORS on IIS(http://enable-cors.org/server_iis7.html)

Step 4: Configure report with your own data

Now it’s time to configure Flexmonster Pivot Component on the web page. Let’s create a minimal report for this (replace filename and other parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "ocsv",
   /* URL to the Data Compressor .NET Core */
```

```
    filename: "http://localhost:55772/api/flexmonster/get"
   }
  },
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

Launch the web page from browser — here you go! A pivot table is embedded into your project.

Examples

Saving to file

```
string connetionString = "server=localhost;database=XXX;";
string sql = "SELECT * FROM YYY";
MySqlConnection sqlConnection = new MySqlConnection(connetionString);
sqlConnection.Open();
MySqlCommand command = new MySqlCommand(sql, sqlConnection);
DbDataReader dataReader = command.ExecuteReader();

Stream inputStream = Compressor.CompressDb(dataReader);
FileStream fileStream = File.Create("data.ocsv");
inputStream.CopyTo(fileStream);
fileStream.Dispose();
```

4.3. Connecting to relational database with Java

Requirements

- Flexmonster Pivot Component version 2.213 or higher
- Java 7 or higher
- Driver for the database

Supported databases

- Oracle Database – driver(http://www.oracle.com/technetwork/database/features/jdbc/index.html)
- MySQL – driver(http://dev.mysql.com/downloads/connector/j/)
- Microsoft SQL Server – driver(http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=11774)
- PostgreSQL – driver(http://jdbc.postgresql.org/download.html)
- Other JDBC databases(http://www.oracle.com/technetwork/java/index-136695.html)

Step 1: Embed the component into your web page

You already have in your HTML page an empty component. If Flexmonster Component is not embedded – return

to Quickstart(/doc/how-to-create-js-pivottable/)
. Your code should look like the following example:

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Step 2: Setup Data Compressor on the server

First of all, you need to add the following dependencies to your project:

- Appropriate database driver
- flexmonster-compressor.jar – it is located in the Download Package(/download-page)

Below is the connection and query sample for MySQL database:

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
String connectionString = "jdbc:mysql://localhost:3306/foodmart";
Connection connection =
 DriverManager.getConnection(connectionString, "user", "pass");
String query = "SELECT * FROM customer";
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(query);
```

The following line of code will convert ResultSet to compressed InputStream:

```
InputStream inputStream = Compressor.compressDb(resultSet);
```

Now, you can create a response from the InputStream. For simple cases, it is enough to read all content:

```
Scanner s = new Scanner(inputStream).useDelimiter("\\A");
String output = s.hasNext() ? s.next() : "";
```

Also, it is possible to create streaming response. It means that the end user will get the response almost without delay and server will use less amount of memory. It is recommended way for large datasets:

```
response.setContentType("text/plain");
OutputStream outputStream = response.getOutputStream();
int length = 0;
```

```
byte[] buffer = new byte[10240];
while ((length = inputStream.read(buffer)) > 0) {
    outputStream.write(buffer, 0, length);
    outputStream.flush();
}
```

Full project is available at the server/java/ inside Download Package(/download-page)
.

Step 3: Enable cross-origin resource sharing (CORS).

By default, browser prevents JavaScript from making requests across domain boundaries. CORS allows web applications to make cross-domain requests. Here are some useful links explaining how to setup CORS on your server:

- **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 report with your own data

Now it’s time to configure Flexmonster Pivot Component on the web page. Let’s create a minimal report for this (replace filename and other parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "ocsv",
   /* URL to the Data Compressor Java */
   filename: "http://localhost:8400/FlexmonsterCompressor/get"
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

Launch the web page from browser — here you go! A pivot table is embedded into your project. Check the example on JSFiddle(http://jsfiddle.net/flexmonster/y4jnrk1y/)
.

Examples

Saving to file

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

```
String connectionString = "jdbc:mysql://localhost:3306/foodmart";
Connection connection =
 DriverManager.getConnection(connectionString, "user", "pass");
String query = "SELECT * FROM customer";
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(query);

InputStream inputStream = Compressor.compressDb(resultSet);
OutputStream outputStream = new FileOutputStream("data.ocsv");
int length = 0;
byte[] buffer = new byte[10240];
while ((length = inputStream.read(buffer)) > 0) {
    outputStream.write(buffer, 0, length);
}
inputStream.close();
outputStream.close();
```

4.4. Connecting to relational database with PHP

Requirements

- Flexmonster Pivot Component version 2.213 or higher
- PHP 5.3+ (earlier versions may work, but not tested)

Supported databases

- MySQL
- Oracle Database
- Microsoft SQL Server
- PostgreSQL
- PDO databases(http://php.net/manual/en/pdo.drivers.php)

- Other databases supported by PHP(http://php.net/manual/en/refs.database.php)

Step 1: Embed the component into your web page

You already have in your HTML page an empty component. If Flexmonster Component is not embedded – return to Quickstart(/doc/how-to-create-js-pivottable/)
. Your code should look like the following example:

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Step 2: Setup Data Compressor on the server

First of all, you need to add the following dependency to your project, flexmonster-compressor.php is located in the Download Package(/download-page)
:

```
require_once("flexmonster-compressor.php");
```

Below is the connection and query sample for MySQL database:

```
mysql_connect($server, $username, $password);
mysql_select_db($dbname);
$result = mysql_query("SELECT * FROM customer");
```

The following line of code will start streaming compressed $result:

```
header('Content-Type: text/plain');
Compressor::compressMySql($result);
```

Full project is available at the server/php/ inside Download Package(/download-page)
.

Step 3: Enable cross-origin resource sharing (CORS).

By default, browser prevents JavaScript from making requests across domain boundaries. CORS allows web applications to make cross-domain requests. Here are some useful links explaining how to setup CORS on your server:

- IIS – CORS on IIS(http://enable-cors.org/server_iis7.html)
- Apache – CORS on Apache(http://enable-cors.org/server_apache.html)

Also, you can add the following headers at the beginning of PHP file:

```
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods', 'OPTIONS,GET,PUT,POST,DELETE');
header('Access-Control-Allow-Headers', 'Content-Type');
```

Step 4: Configure report with your own data

Now it’s time to configure Flexmonster Pivot Component on the web page. Let’s create a minimal report for this (replace filename and other parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "ocsv",
   /* URL to the Data Compressor PHP */
   filename: "http://localhost/demo-compress-mysql.php"
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

Launch the web page from browser — here you go! A pivot table is embedded into your project. Check the example on JSFiddle(http://jsfiddle.net/flexmonster/y4jnrk1y/)
.

Examples

Here you can find few PHP examples of compressing different databases.

MySQL

```
header("Content-Type: text/plain");
require_once("flexmonster-compressor.php");

mysql_connect("localhost", "username", "password");
mysql_select_db("foodmart");
$result = mysql_query("SELECT * FROM customer");
Compressor::compressMySql($result);
```

MySQLi

```
header("Content-Type: text/plain");
require_once("flexmonster-compressor.php");

$mysqli = new mysqli("localhost", "username", "password", "foodmart");
$result = $mysqli->query("SELECT * FROM customer");
Compressor::compressMySqli($result);
```

Microsoft SQL Server

```
header("Content-Type: text/plain");
require_once("flexmonster-compressor.php");

$conn = sqlsrv_connect("localhost", array(
    "Database"=>"foodmart",
    "UID"=>"username",
    "PWD"=>"password"
  )
);
$result = sqlsrv_query($conn, "SELECT * FROM customer");
Compressor::compressSQLSRV($result);
```

PostgreSQL

```
header("Content-Type: text/plain");
require_once("flexmonster-compressor.php");

$conn = pg_connect("host=localhost dbname=foodmart user=username password=password")
;
$result = pg_query($conn, "SELECT * FROM customer");
Compressor::compressPostgreSQL($result);
```

Oracle Database

```
header("Content-Type: text/plain");
require_once("flexmonster-compressor.php");

$conn = oci_connect("username", "password", "localhost/XE");
$stid = oci_parse($conn, 'SELECT * FROM customer');
oci_execute($stid);
Compressor::compressOCI($stid);
```

PDO / MySQL example

```
header("Content-Type: text/plain");
require_once("flexmonster-compressor.php");

$conn = new PDO("mysql:host=localhost;dbname=foodmart", "username", "password");
$result = $conn->query("SELECT * from customer");
Compressor::compressPDO($result);
```

Other Databases / MySQL example

```
header("Content-Type: text/plain");
require_once("flexmonster-compressor.php");

mysql_connect("localhost", "username", "password");
mysql_select_db("foodmart");
$result = mysql_query("SELECT * FROM customer");

$header = array();
for ($i=0; $i < mysql_num_fields($result); $i++) {
  $header[$i] = array(
    'name' => mysql_field_name($result, $i),
    'type' => mysql_field_type($result, $i)
  );
}
$array = array();
$array[] = $header;
while ($row = mysql_fetch_row($result)) {
  $array[] = $row;
}
Compressor::compressArray($array);
```

Saving to file / MySQL example

```
header("Content-Type: text/plain");
require_once("flexmonster-compressor.php");

mysql_connect("localhost", "username", "password");
mysql_select_db("foodmart");
mysql_set_charset("utf8");
$result = mysql_query("SELECT * FROM customer");
$outFile = "data.ocsv";
Compressor::compressMySql($result, $outFile);
```

Troubleshooting

Data is not streaming

It is likely that output_buffering is enabled in the PHP configuration on your server and therefore data is not streaming. Please try to disable it in the php.ini for better performance:
output_buffering = Off (see
http://php.net/manual/en/outcontrol.configuration.php(http://php.net/manual/en/outcontrol.configuration.php)
).

5. Connecting to Microsoft Analysis Services

There are two ways to connect to Microsoft Analysis Services using Flexmonster Pivot Table:

1. via XMLA(/doc/connecting-to-microsoft-analysis-services/#xmla)
   – an industry standard for data access in analytical systems
2. via Flexmonster Accelerator(/doc/connecting-to-microsoft-analysis-services/#accelerator)
   – special server-side utility developed by Flexmonster

If you already have configured XMLA (msmdpump.dll) it will be preferable to start with option #1. In case you do not have XMLA or you need some advanced features (increase loading speed, use credentials, etc.) – option #2 is a better choice.

Connecting to MS Analysis Services via XMLA

XMLA (XML for Analysis) – an industry standard for data access in analytical systems, such as OLAP and Data Mining. Please follow the steps below to configure a connection to Microsoft Analysis Services via XMLA.

Step 1: Embed the component into your web page

You already have in your HTML page an empty component. If Flexmonster Component is not embedded – return to Quickstart(/doc/how-to-create-js-pivottable/)
. Your code should look like the following example:

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Step 2: Configure XMLA access to the cube

If you have XMLA already configured please skip this step. Otherwise, please refer to the article that explains how to set up an HTTP endpoint for accessing an Analysis Services instance(http://msdn.microsoft.com/en-us/library/gg492140.aspx)
.

Step 3: Enable cross-origin resource sharing (CORS)

By default, browser prevents JavaScript from making requests across domain boundaries. CORS allows web applications to make cross-domain requests. Please read this step-by-step instruction to enable CORS for IIS(/question/stream-error-occurred-while-loading-httplocalhost8080olapmsmdpump-dll)
to read data from Microsoft Analysis Services.

Step 4: Configure report with your own data

Now it's time to configure Flexmonster Pivot Component on the web page. Let's create a minimal report for this (replace proxyUrl, catalog and cube parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "microsoft analysis services",

   /* URL to msmdpump.dll */
   proxyUrl: "http://olap.flexmonster.com/olap/msmdpump.dll",

   /* Catalog name */
   catalog: "Adventure Works DW Standard Edition",

   /* Cube name */
   cube: "Adventure Works",
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

Launch the web page from browser — here you go! A pivot table is embedded into your project. Check out the example on JSFiddle(http://jsfiddle.net/flexmonster/3z3abpfs/)
.

Optimize data loading

subquery parameter helps Flexmonster component to take less time for loading and rendering. Below is an example of showing reports for the specific year from the date hierarchy:

```
{
    "dataSource": {
        "dataSourceType": "microsoft analysis services",
        "proxyUrl": "http://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 on JSFiddle(http://jsfiddle.net/flexmonster/3dmbzyp5/)
.

Connecting to MS Analysis Services via Flexmonster Accelerator

Everyone who has ever worked with multidimensional databases analysis has most probably faced the problem of finding a suitable component that not only performs all required functions but also executes them fast. Flexmonster solved this problem and created **Flexmonster Accelerator for Microsoft Analysis Services** cubes – a special server-side proxy that helps you to increase data loading speed from server to customer's browser.

Working with OLAP cubes, a browser component is communicating with the server via XMLA protocol. It's no secret that the XMLA protocol is heavy and exchanges a lot of excessive information. Thus, it takes too much time and memory to load and process the data.

We replaced XMLA protocol and use direct requests from the Component to a server.

In this way, we have come up with solutions to two major problems that many of those who work with big data had faced to a different extent:

- We made big data transfer from server to browser enormously fast. Our tool allows you to transfer large multidimensional data in super easy and fast way. Reporting becomes more enjoyable and prompt for your end users.
- We greatly reduced the load on a browser memory.

Please refer to Getting started with Accelerator(/doc/getting-started-with-accelerator-ssas/)
guide to find step-by-step instructions.

5.1. MS Analysis Services / Getting started with Accelerator

Everyone who has ever worked with multidimensional databases analysis has most probably faced the problem of finding a suitable component that not only performs all required functions but also executes them fast. Flexmonster solved this problem and created **Flexmonster Accelerator for Microsoft Analysis Services** cubes – a special server-side proxy that helps you to increase data loading speed from server to customer's browser.

Working with OLAP cubes, a browser component is communicating with the server via XMLA protocol. It's no secret that the XMLA protocol is heavy and exchanges a lot of excessive information. Thus, it takes too much time and memory to load and process the data.

We replaced XMLA protocol and use direct requests from the Component to a server.

In this way, we have come up with solutions to two major problems that many of those who work with big data had faced to a different extent:

- We made big data transfer from server to browser enormously fast. Our tool allows you to transfer large multidimensional data in super easy and fast way. Reporting becomes more enjoyable and prompt for your end users.
- We greatly reduced the load on a browser memory.

Open Live Demo(/demos/connect-msas)

Requirements

- Flexmonster Pivot Component version 2.2 or higher
- Microsoft Analysis Services installed and configured
- Microsoft .NET Framework 4 or higher

Step 1: Embed the component into your web page

You already have in your HTML page an empty component. If Flexmonster Component is not embedded – return to Quickstart(/doc/how-to-create-js-pivottable/)
. Your code should look like the following example:

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Step 2: Configure Data Speed Accelerator on the server

The package contains following files:

- flexmonster-proxy-ssas.exe – server-side utility that handles connectivity between Microsoft Analysis Services and Flexmonster Pivot Component
- flexmonster.config – file that contains configuration parameters for the utility (connection string, port, etc.)

First of all, let’s review flexmonster.config file. It contains following parameters:

- CONNECTION_STRING – connection string for Microsoft Analysis Services. Example: *Data Source=localhost;*. Required.
- PORT – port number for the proxy service endpoint. Optional. The default is 50005.
- CACHE_MEMORY_LIMIT – size of maximum RAM memory available for cache (in MB). Optional. The default is 0 (unlimited).
- CACHE_ENABLED – indicates whether the cache is enabled. Optional. The default is true. Available since version 2.211.

After configuring all the necessary options, Data Speed Accelerator is ready to be launched. Just run the flexmonster-proxy-ssas.exe with Administrator privileges.

You can check the Accelerator is up and running by navigating to its URL in the browser (i.e. http://localhost:50005(http://localhost:50005/)
).

Step 3: Open a port for Data Speed Accelerator in the firewall

If it's planned to allow connection to Data Speed Accelerator from outside of the server, you should open an appropriate port in the firewall. Default port number is 50005, but it may vary depending on PORT parameter in flexmonster.config file.

Step 4: Configure Flexmonster Pivot Component

Now it's time to configure the client – Flexmonster Pivot Component. Let's create a minimal configuration using JavaScript API (replace proxyUrl, catalog and cube parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "microsoft analysis services",

   /* URL to the Data Speed Accelerator */
   proxyUrl: "http://localhost:50005",

   /* Catalog name */
   catalog: "Adventure Works DW Standard Edition",

   /* Cube name */
   cube: "Adventure Works",

   // Flag to use Data Speed Accelerator instead of XMLA protocol
   binary: true
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

Launch the web page from browser — here you go! A pivot table is embedded into your project.

Cache control

Usually, the cache is a great help, but the cache becomes out of date if the underlying database is changing. By default, cache is enabled and controlled by the Accelerator.

Also, it is possible to disable cache by the following parameter in flexmonster.config (available since version 2.211):

```
CACHE_ENABLED = false
```

5.2. Installing Accelerator as a Windows Service

Overview

There is an option to install Flexmonster Accelerator as a Windows Service instead of using Console application. Main benefits of running Accelerator as a Windows Service are:

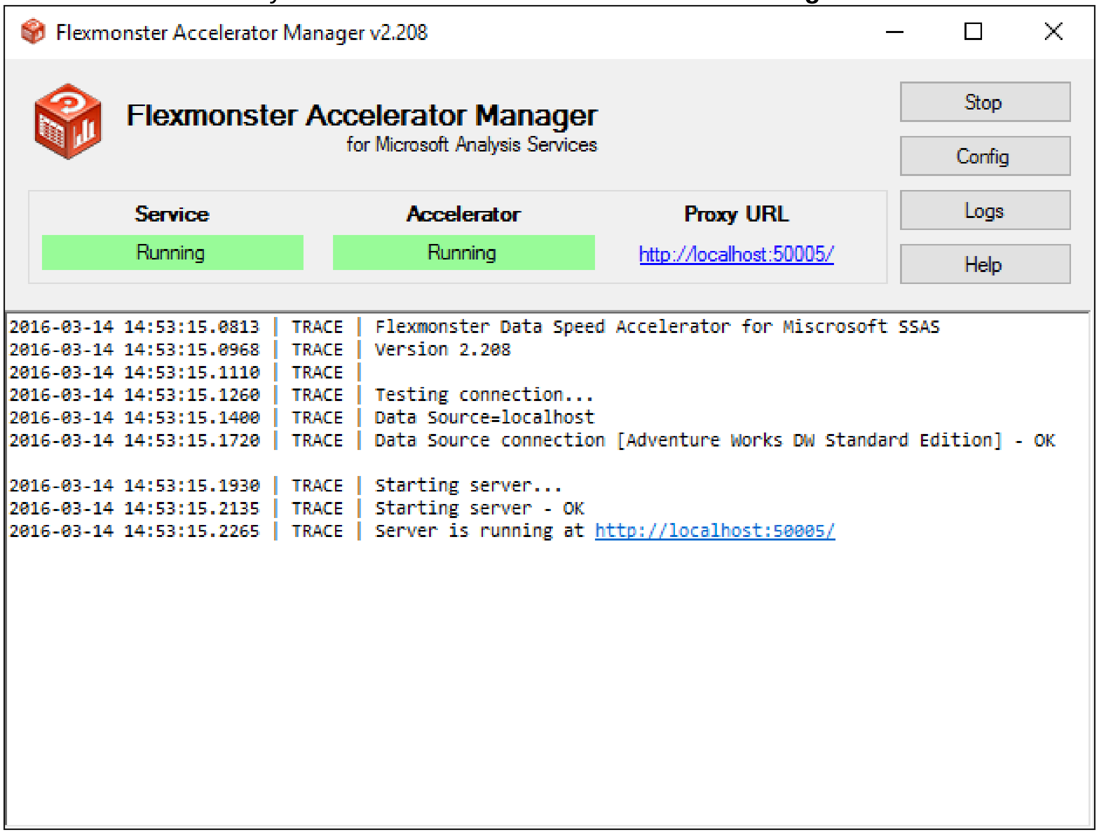
- it's running in the background and out of sight
- it's starting automatically on Windows startup
- it's harder for a user to inadvertently quit the application

Requirements

- Flexmonster Pivot Component version 2.2 or higher
- Microsoft Analysis Services installed and configured
- Microsoft .NET Framework 4 or higher

Step 1: Configure Data Speed Accelerator on the server

Start installation using server/installer/Flexmonster Accelerator.msi setup file and follow the wizard. After successful installation you can 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/
```

First of all, let's review Config file. It contains following parameters:

- CONNECTION_STRING – connection string for Microsoft Analysis Services. Example: Data Source=localhost;. Required.
- PORT – port number for the proxy service endpoint. Optional. Default is 50005.
- CACHE_MEMORY_LIMIT – size of maximum RAM memory available for cache (in MB). Optional. Default

is 0 (unlimited).
- CACHE_ENABLED – indicates whether the cache is enabled. Optional. Default is true. Available since version 2.211.

You can check the Accelerator is up and running by navigating to its URL in the browser (i.e. http://localhost:50005(http://localhost:50005/)
).

After configuration you can close **Flexmonster Accelerator Manager** – service will continue working in the background.

Step 2: Open a port for Data Speed Accelerator in the firewall

If it's planned to allow connection to Data Speed Accelerator from outside of the server, you should open an appropriate port in the firewall. Default port number is 50005, but it may vary depending on PORT parameter in flexmonster.config file.

Step 3: Configure Flexmonster Pivot Component

Now it's time to configure the client – Flexmonster Pivot Component. Let's create a minimal configuration using JavaScript API (replace proxyUrl, catalog and cube parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "microsoft analysis services",

   /* URL to the Data Speed Accelerator */
   proxyUrl: "http://localhost:50005",

   /* Catalog name */
   catalog: "Adventure Works DW Standard Edition",

   /* Cube name */
   cube: "Adventure Works",

   // Flag to use Data Speed Accelerator instead of XMLA protocol
   binary: true
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

5.3. MS Analysis Services / Configuring username/password protection

Overview

Flexmonster Pivot Component supports credentials to access the data source. It is commonly used for the following purposes:

- provide credentials for data source connectivity

- provide access for users with different roles and access levels

Here are the step-by-step instructions to configure secure connection with username/password protection and define different usernames on the client side. If you connect to the cube via XMLA you can use roles from MSAS(#roles)
.

Requirements

- Flexmonster Pivot Component version 2.205 or higher
- Microsoft Analysis Services installed and configured
- Microsoft .NET Framework 4 or higher

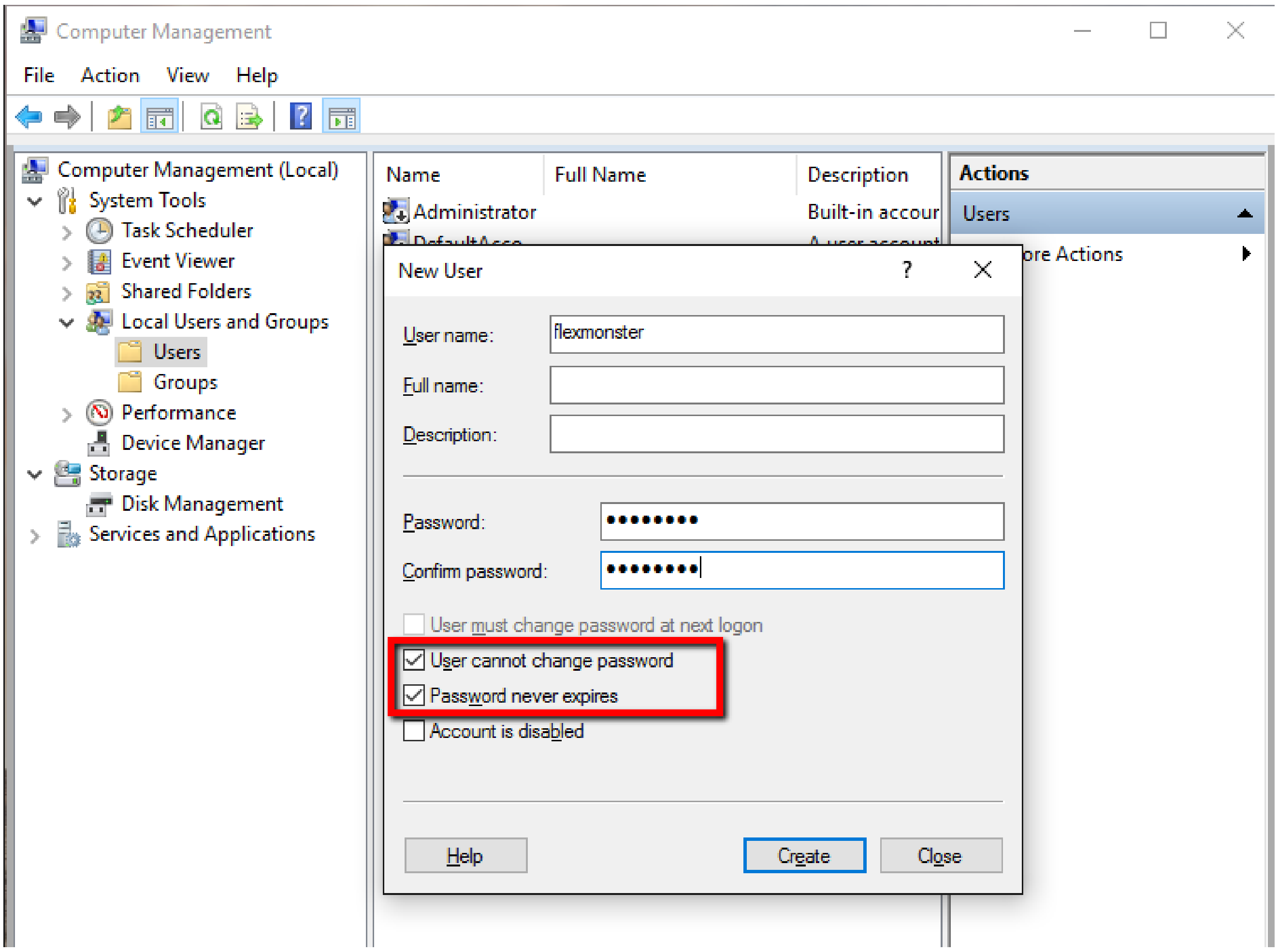
Step 1: Configure secure HTTP endpoint for accessing an Analysis Services

First of all, you need to set up HTTP endpoint for Analysis Services on IIS server. If it is already done, skip this step. Regarding IIS configuration, please refer to the MSDN documentation(http://msdn.microsoft.com/en-us/library/gg492140(v=sql.120).aspx)
. In this step, you also configure authentication types(http://msdn.microsoft.com/en-us/library/gg492140(v=sql.120).aspx#Anchor_4)
.

Step 2: Create a default user for Accelerator

One username/password is used in the flexmonster.config file for the Accelerator on the server side. In general, Accelerator will use these credentials if client-side report does not contain own credentials. Also, these credentials are used to start the Accelerator, check connection to the data source and connect anonymous users. So, it is recommended to create a user for Accelerator with default privileges and DO NOT use an admin account for this purpose.

Navigate to **Control Panel > Administrative Tools > Computer Management** and choose **System Tools > Local Users and Groups > Users**. Add a new user (i.e. flexmonster) by right-click as shown in the screenshot.



New User

User name: flexmonster
Full name:
Description:
Password: ••••••••
Confirm password: ••••••••

☐ User must change password at next logon
☑ User cannot change password
☑ Password never expires
☐ Account is disabled

Step 3: Configure Accelerator

Open flexmonster.config and specify following CONNECTION_STRING parameters:

```
CONNECTION_STRING=Data Source=<endpoint_url>;UID=<username>;Password=<password>;
```

Where:

- <endpoint_url> – URL to HTTP endpoint (i.e. http://localhost/ssas/msmdpump.dll)
- <username> – username of the default user (i.e. flexmonster)
- <password> – password of the default user

Accelerator is ready to be launched and should successfully connect to the data source. Just run the flexmonster-proxy-ssas.exe with Administrator privileges.

You can check the Accelerator is up and running by navigating to its URL in the browser (i.e. http://localhost:50005(http://localhost:50005/)
).

Configuring MSAS roles

Use roles from MSAS to manage permissions for different users. Have a look at the following sample (replace proxyUrl, catalog, cube and roles parameters with your specific values):

```
{
    dataSource: {
        dataSourceType: "microsoft analysis services",
  // URL to msmdpump.dll
        proxyUrl: "http://olap.flexmonster.com/olap/msmdpump.dll",
        catalog: "Adventure Works DW Standard Edition",
        cube: "Adventure Works",
        // roles from MSAS (only via XMLA protocol)
        roles: "admin,manager"
    }
}
```

5.4. MS Analysis Services / Configuring secure HTTPS connection

Overview

All data that is sent by HTTP is not encrypted and can be intercepted. That’s why we have added an option to enable HTTPS for Accelerator. HTTPS encrypts all data that is sent from the client to Accelerator and vice versa. Please follow the steps below to configure secure HTTPS connection for your setup.

Requirements

- Flexmonster Pivot Component version 2.206 or higher
- Microsoft Analysis Services installed and configured
- Microsoft .NET Framework 4 or higher
- **Valid and trusted SSL certificate**

Step 1: Register the server certificate

On an elevated console (“Run as administrator”), register the server certificate by running the following command:

```
netsh http add sslcert ipport=0.0.0.0:<port> certhash=<hash> appid=<app-guid>
```

Where:

- <port> – the listening port (i.e. 50005); the special IP address 0.0.0.0 matches any IP address for the local machine
- <hash> – the certificate’s SHA-1 hash, represented in hexadecimal
- <app-guid> – any GUID (i.e. {00000000-0000-0000-0000-000000000000}), used to identity the owning application

In case of any errors, please verify that the SSL certificate is properly installed in the Personal store and contains private key assigned.
Aslo, please note that appid parameter contains single quotes and curly braces ('{}').

Step 2: Enable HTTPS in Accelerator

After you have the certificate registered let’s enable the HTTPS in the Accelerator’s config. Open flexmonster.config file and modify/add the HTTPS parameter as follows:

```
HTTPS=true
```

Available values for HTTPS parameter is true or false. By default HTTPS is disabled (false).

Data Speed Accelerator is ready to be launched. Just run the flexmonster-proxy-ssas.exe with Administrator privileges.

You can check the Accelerator is up and running by navigating to its URL in the browser (i.e. https://localhost:50005(https://localhost:50005/)
).

Step 3: Configure Flexmonster Pivot Component

Now it’s time to configure the client – Flexmonster Pivot Component. Let’s create a minimal configuration using JavaScript API (replace proxyUrl, catalog and cube parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "microsoft analysis services",
   proxyUrl: "https://localhost:50005",
   catalog: "Adventure Works DW Standard Edition",
   cube: "Adventure Works",
   binary: true
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

Please note that proxyUrl now contains https://. That means that the data is protected and encrypted with your SSL certificate.

5.5. MS Analysis Services / Troubleshooting

Erorr: Your current version of Flexmonster Pivot Component is not compatible with Data Speed Accelerator. Please update the component to the minimum required version: X.XXX

It means that versions of Flexmonster Accelerator and Pivot Table Component are different and not compatible. To fix this error it is recommended to update both items to the latest available version. Please refer to the “How to update” guide(/doc/updating-to-the-latest-version/)
for more details.

Error: Your current version of Data Speed Accelerator is not compatible with Flexmonster Pivot Component. Please update the Accelerator to the minimum required version: X.XXX

It means that versions of Flexmonster Accelerator and Pivot Table Component are different and not compatible. To fix this error it is recommended to update both items to the latest available version. Please refer to the "How to update" guide(/doc/updating-to-the-latest-version/)
for more details.

6. Connecting to Pentaho Mondrian

There are two ways to connect to Pentaho Mondrian using Flexmonster Pivot Table:

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)
   – special server-side utility developed by Flexmonster

If you already have configured XMLA provider it will be preferable to start with option #1. In case you do not have XMLA or you need some advanced features (increase loading speed, use credentials, etc.) – option #2 is a better choice.

Connecting to Pentaho Mondrian via XMLA

XMLA (XML for Analysis) – an industry standard for data access in analytical systems, such as OLAP and Data Mining. Please follow the steps below to configure a connection to Pentaho Mondrian via XMLA.

Step 1: Embed the component into your web page

You already have in your HTML page an empty component. If Flexmonster Component is not embedded – return to Quickstart(/doc/how-to-create-js-pivottable/)
. Your code should look like the following example:

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Step 2: Configure XMLA access to the cube

If you have XMLA already configured please skip this step. Otherwise, please refer to the 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, browser prevents JavaScript from making requests across domain boundaries. CORS allows web applications to make cross-domain requests. Please find instructions for common Java servers below:

- 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 report with your own data

Now it's time to configure Flexmonster Pivot Component 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 = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "mondrian",

   /* Data Source Info */
   dataSourceInfo: "MondrianFoodMart",

   /* URL to XMLA provider */
   proxyUrl: "http://olap.flexmonster.com:8080/mondrian/xmla",

   /* Catalog name */
   catalog: "FoodMart",

   /* Cube name */
   cube: "Sales"
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

Launch the web page from browser — here you go! A pivot table is embedded into your project. Try the example on JSFiddle(http://jsfiddle.net/flexmonster/uf4tpdz9/)
.

Connecting to Pentaho Mondrian via Flexmonster Accelerator

Everyone who has ever worked with multidimensional databases analysis has most probably faced the problem of finding a suitable component that not only performs all required functions but also executes them fast. Flexmonster solved this problem and created **Flexmonster Accelerator for Pentaho Mondrian** cubes – a special server-side proxy that helps you to increase data loading speed from server to customer's browser.

Working with OLAP cubes, a browser component is communicating with the server via XMLA protocol. It's no secret that the XMLA protocol is heavy and exchanges a lot of excessive information. Thus, it takes too much time and memory to load and process the data.

We replaced XMLA protocol and use direct requests from the Component to a server.

In this way, we have come up with solutions to two major problems that many of those who work with big data had faced to a different extent:

- We made big data transfer from server to browser enormously fast. Our tool allows you to transfer large multidimensional data in super easy and fast way. Reporting becomes more enjoyable and prompt for your end users.
- We greatly reduced the load on a browser memory.

Please refer to Getting started with Accelerator(/doc/getting-started-with-accelerator-mondrian/)
guide to find step-by-step instructions.

6.1. Mondrian / Getting started with Accelerator

Everyone who has ever worked with multidimensional databases analysis has most probably faced the problem of finding a suitable component that not only performs all required functions but also executes them fast. Flexmonster solved this problem and created **Flexmonster Accelerator for Pentaho Mondrian** cubes – a special server-side proxy that helps you to increase data loading speed from server to customer's browser.

Working with OLAP cubes, a browser component is communicating with the server via XMLA protocol. It's no secret that the XMLA protocol is heavy and exchanges a lot of excessive information. Thus, it takes too much time and memory to load and process the data.

We replaced XMLA protocol and use direct requests from the Component to a server.

In this way, we have come up with solutions to two major problems that many of those who work with big data had faced to a different extent:

- We made big data transfer from server to browser enormously fast. Our tool allows you to transfer large multidimensional data in super easy and fast way. Reporting becomes more enjoyable and prompt for your end users.
- We greatly reduced the load on a browser memory.

Open Live Demo(/demos/connect-mondrian)

Requirements

- Flexmonster Pivot Component version 2.2 or higher
- Relational database and Mondrian schema for it
- Java JRE 1.7+

Step 1: Embed the component into your web page

You already have in your HTML page an empty component. If Flexmonster Component is not embedded – return to Quickstart(/doc/how-to-create-js-pivottable/)
. Your code should look like the following example:

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

```
<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Step 2: Configure Data Speed Accelerator on the server

The package contains following files (server/ folder):

- flexmonster-proxy-mondrian.jar – server-side utility that handles connectivity between Pentaho Mondrian and Flexmonster Pivot Component
- flexmonster-proxy-mondrian-4.jar – same as flexmonster-proxy-mondrian.jar, but for newer version of Pentaho Mondrian 4.4
- flexmonster.config – file that contains configuration parameters for the utility (connection string, port, etc.)

Also, there are additional sample files:

- FoodMart.xml – sample Mondrian schema for FoodMart database
- FoodMart-4.xml – same as FoodMart.xml, but for newer version of Pentaho Mondrian 4.4
- mysql-connector-java-*.jar – Java connector for MySQL database

First of all, let’s review flexmonster.config file. It contains following parameters:

- CONNECTION_STRING – connection string for Pentaho Mondrian. Required.
  Example:
  Jdbc=jdbc:mysql://localhost:3306/foodmart?user=root&password=password;JdbcDrivers=com.mysql.jdbc.Driver;
- JDBC_DRIVER_JAR – path to the Java connector (.jar file) for database. Required.
  Example: *./mysql-connector-java-5.1.37.jar*
- JDBC_DRIVER_CLASS – class name of the Java connector. Required.
  Example: *com.mysql.jdbc.Driver*
- CATALOGS_PATH – a path to the folder that contains Mondrian schemas. Optional. The default is ./
- PORT – port number for the proxy service endpoint. Optional. The default is 50006
- CACHE_MEMORY_LIMIT – size of maximum RAM memory available for cache (in MB). Optional. The default is 0 (unlimited).
- CACHE_ENABLED – indicates whether the cache is enabled. Optional. The default is true. Available since version 2.211.

After configuring all the necessary options, Data Speed Accelerator is ready to be launched. Just execute the following commands in terminal:

```
# navigate to folder that contains Accelerator
cd {path_to_package}/server
# start Accelerator
java -jar flexmonster-proxy-mondrian.jar
```

If your schema is built for Mondrian 4, just use flexmonster-proxy-mondrian-4.jar instead.

Step 3: Open a port for Data Speed Accelerator in the firewall

If it’s planned to allow connection to Data Speed Accelerator from outside of the server, you should open an appropriate port in the firewall. Default port number is 50006, but it may vary depending on PORT parameter in flexmonster.config file.

Step 4: Configure Flexmonster Pivot Component

Now it’s time to configure the client – Flexmonster Pivot Component. Let’s create a minimal configuration using JavaScript API (replace proxyUrl, catalog and cube parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "mondrian",

   /* URL to the Data Speed Accelerator */
   proxyUrl: "http://localhost:50006",

   /* Data Source Info */
   dataSourceInfo: "MondrianFoodMart",

   /* Catalog name / Mondrian schema */
   catalog: "FoodMart",

   /* Cube name */
   cube: "Sales",

   // Flag to use Data Speed Accelerator instead of XMLA protocol
   binary: true
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

Launch the web page from browser — here you go! A pivot table is embedded into your project.

Cache control

Accelerator for Mondrian has two levels of cache:

- Mondrian cache
- Accelerator cache (controlled by CACHE_ENABLED parameter)

Usually, the cache is a great help, but the cache becomes out of date if the underlying database is changing. Mondrian cannot deduce when the database is being modified, so we introduce a method to force clearing the Mondrian cache.

It works by triggering the “ClearCache” as follows:

```
http://localhost:50006/FlexmonsterProxy/ClearCache(http://localhost:50006/Flexmonste
rProxy/ClearCache)
```

Accelerator cache can be disabled by the following 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 executes. It is possible to use any of these properties with Accelerator.

You just need to create mondrian.properties file in the same location as flexmonster-proxy-mondrian.jar.

For example:

```
mondrian.rolap.aggregates.ChooseByVolume = false
mondrian.rolap.aggregates.generateSql = false
```

Please refer to the full list of Mondrian properties(http://mondrian.pentaho.com/documentation/configuration.php) to find out more.

6.2. Configuring Mondrian roles

Overview

Sometimes it’s necessary to limit access to some parts of the data. To solve 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 Component.

Requirements

- Flexmonster Pivot Component version 2.210 or higher
- Relational database and Mondrian schema for it
- Java JRE 1.7+

Step 1: Configure roles in Mondrian schema file

First of all, you need to configure roles in the Mondrian schema file. Please refer to the Mondrian documentation(http://mondrian.pentaho.com/documentation/schema.php#Access_control)
for more details.

Step 2: Launch Flexmonster Accelerator

Start (or restart) Flexmonster Accelerator for Mondrian. Please refer to the Accelerator “Getting Started” guide(/doc/getting-started-with-accelerator-mondrian/)
for more details.

Step 3: Configure Flexmonster Pivot Component

Now, let’s specify Mondrian roles in the configuration. Here is a minimal sample created with JavaScript API (replaceproxyUrl, catalog, cube and roles parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "mondrian",
   proxyUrl: "localhost:50006",
   dataSourceInfo: "MondrianFoodMart",
   catalog: "FoodMart",
   cube: "Sales",
   binary: true,
   // Mondrian roles
   roles: "California manager"
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

6.3. Mondrian / ?onfiguring username/password protection

Overview

Flexmonster Pivot Component supports credentials to access the data source. It is commonly used for the following purposes:

- provide credentials for data source connectivity
- provide access for users with different roles and access levels

Here are the step-by-step instructions to configure secure connection with username/password protection.

Requirements

- Flexmonster Pivot Component version 2.205 or higher
- Relational database and Mondrian schema for it
- Java JRE 1.7+

Step 1: Create a default user for Accelerator

We recommend creating a default user for Accelerator with default privileges. It is used to start the Accelerator, check connection to the data source and connect anonymous users. Refer to the documentation of your database to create a new user (i.e. MySQL(http://dev.mysql.com/doc/refman/5.7/en/adding-users.html)
, Oracle(http://docs.oracle.com/cd/B19306_01/network.102/b14266/admusers.htm)
).

Step 2: Configure Accelerator

Open flexmonster.config and specify user and password in the CONNECTION_STRING parameters. After editing, flexmonster.config should look like the following:

```
CONNECTION_STRING=Jdbc=jdbc:mysql://localhost:3306/foodmart?user=flexmonster&passwor
d=password;JdbcDrivers=com.mysql.jdbc.Driver;
```

Accelerator is ready to be launched. Just execute the following command in terminal:

```
java -jar flexmonster-proxy-mondrian.jar
```

You can check the Accelerator is up and running by navigating to its URL in the browser (i.e.
http://localhost:50006(http://localhost:50006/)
).

6.4. Mondrian / ?onfiguring secure HTTPS connection

Overview

All data that is sent by HTTP is not encrypted and can be intercepted. That’s why we have added an option to enable HTTPS for Accelerator. HTTPS encrypts all data that is sent from the client to Accelerator and vice versa. Please follow the steps below to configure secure HTTPS connection for your setup.

Requirements

- Flexmonster Pivot Component version 2.205 or higher
- Relational database and Mondrian schema for it
- Java JRE 1.7+
- **Valid and trusted SSL certificate**

Step 1: Import 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 the following:

```
keytool -importkeyst
ore -destkeystore keystore.jks -srckeystore <private_key> -srcstoretype PKCS12
```

Where:

- <private_key> – path to .p12 file with the private key (i.e. *flexmonster.p12*)

It will ask to enter a new password for keystore and a password for the private key. In a result, it will generate a keystore.jks file with imported private key inside. Copy this file to some location, it will be necessary for the next step.

Step 2: Enable HTTPS in Accelerator

After you have the certificate imported let's enable the HTTPS in the Accelerator's config. Open 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 JKS file (i.e. *keystore.jks*)
- <keystore_password> – password for the keystore
- <private_key_password> – password for the imported key

Available values for HTTPS parameter is true or false. By default HTTPS is disabled (false).

Accelerator is ready to be launched. Just execute the following command in terminal:

```
java -jar flexmonster-proxy-mondrian.jar
```

You can check the Accelerator is up and running by navigating to its URL in the browser (i.e. https://localhost:50006(https://localhost:50006/)
).

Step 3: Configure Flexmonster Pivot Component

Now it's time to configure the client – Flexmonster Pivot Component. Let's create a minimal configuration using JavaScript API (replace proxyUrl, catalog and cube parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "mondrian",
   proxyUrl: "https://localhost:50006",
   dataSourceInfo: "MondrianFoodMart",
   catalog: "FoodMart",
   cube: "Sales",
   binary: true
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

Please note that proxyUrl now contains https://. That means that the data is protected and encrypted with your SSL certificate.

6.5. Mondrian / Troubleshooting

Error: Your current version of Flexmonster Pivot Component is not compatible with Data Speed Accelerator. Please update the component to the minimum required version: X.XXX

It means that versions of Flexmonster Accelerator and Pivot Table Component are different and not compatible. To fix this error it is recommended to update both items to the latest available version. Please refer to the “How to update” guide(/doc/updating-to-the-latest-version/)
for more details.

Error: Your current version of Data Speed Accelerator is not compatible with Flexmonster Pivot Component. Please update the Accelerator to the minimum required version: X.XXX

It means that versions of Flexmonster Accelerator and Pivot Table Component are different and not compatible. To fix this error it is recommended to update both items to the latest available version. Please refer to the “How to update” guide(/doc/updating-to-the-latest-version/)
for more details.

Error: java.lang.OutOfMemoryError: GC overhead limit exceeded

Please try to start 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 – that results in an UnknownHostException, often after a lengthy DNS lookup attempt. Here are few possible actions:

- the first suggestion is to ignore the message. Given that error the application is working fine and the message is only seen in the logs. Please try to open your browser and navigate to the following URL: http://127.0.0.1:50006/ or http://localhost:50006/. There should be a message “Flexmonster Data Speed Accelerator for Pentaho Mondrian”.
- or you can try to add the following record to the hosts file: 127.0.0.1 <hostname>

7. Connecting to icCube

Please follow the steps below to configure a connection to icCube via XMLA service.

Step 1: Embed the component into your web page

You already have in your HTML page an empty component. If Flexmonster Component is not embedded – return to Quickstart(/doc/how-to-create-js-pivottable/)
. Your code should look like the following example:

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

```
<script src="flexmonster/lib/jquery.min.js"></script>
<script src="flexmonster/flexmonster.js"></script>

<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Step 2: Configure XMLA access to the cube

Assuming the server with the default configuration is running on the local machine, the XMLA service is available at the following address:

```
XMLA/HTTP : http://localhost:8282/icCube/xmla
```

Please refer to the icCube documentation(http://www.iccube.com/support/documentation/index.php) for more details.

Step 3: Enable cross-origin resource sharing (CORS)

By default, browser prevents JavaScript from making requests across domain boundaries. CORS allows web applications to make cross-domain requests. Please open icCube configuration file (located at $install/bin/icCube.xml) and edit the following:

1. Cross Origin filter:

```
<icCubeConfiguration>
    ...
    <xmlaComponentConfiguration>
        ...
        <filter>Cross Origin</filter>
        ...
    </xmlaComponentConfiguration>
    ...
</icCubeConfiguration>
```

2. Cross Origin filter configuration:

```
<icCubeConfiguration>
    ...
    <filterConfiguration>
        ...
        <filter>
            <filter-name>Cross Origin</filter-name>
            <filter-class>crazydev.iccube.server.crossorigin.IcCubeCrossOrigin
ServletFilter</filter-class>
            <init-param>
                <param-name>allowedOrigins</param-name>
                <param-value>*</param-value>
            </init-param>
            <init-param>
                <param-name>allowedMethods</param-name>
                <param-value>GET,POST,HEAD,OPTIONS</param-value>
            </init-param>
            <init-param>
                <param-name>allowedHeaders</param-name>
                <param-value>X-Requested-With,Content-Type,Accept,Origin,X-
DataSource-Auth</param-value>
            </init-param>
        </filter>
        ...
    </filterConfiguration>
    ...
</icCubeConfiguration>
```

Step 4: Configure report with your own data

Now it’s time to configure Flexmonster Pivot Component on the web page. Let’s create a minimal report for this (replace proxyUrl, catalog and cube parameters with your specific values):

```
var pivot = $("#pivotContainer").flexmonster({
 toolbar: true,
 report: {
  dataSource: {
   dataSourceType: "iccube",

   /* URL to XMLA service */
   proxyUrl: "http://olap.flexmonster.com:8282/icCube/xmla",

   /* Catalog name */
   catalog: "Sales",

   /* Cube name */
   cube: "Sales"
  }
 },
 licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
});
```

Launch the web page from browser — here you go! A pivot table is embedded into your project. Open the example on JSFiddle(http://jsfiddle.net/flexmonster/oz4hqu9t/)
.

Please visit our live icCube demo(/demos/connect-iccube)
to see one more sample.

Optimize data loading

Use subquery parameter to reduce the amount of data loading from the OLAP cube. For example, you have the cube with the geography hierarchy and your task is to show reports only for the one specific region instead of loading all the data. To show reports only for that region add subquery the following way:

```
{
    "dataSource": {
        "dataSourceType": "iccube",
        "proxyUrl": "http://olap.flexmonster.com:8282/icCube/xmla",
        "catalog": "Sales",
        "cube": "Sales",
        "subquery": "select {[Customers].[Geography].[Region].&[EU]}
                on columns from [Sales]"
    },
    "slice": {
        "rows": [ { "uniqueName": "[Customers].[Geography]" } ],
        "columns": [
            { "uniqueName": "[Product].[Product]" },
            { "uniqueName": "[Measures]" }
        ],
        "measures": [ { "uniqueName": "[Measures].[Amount]" } ]
    }
}
```

Open the example on JSFiddle(http://jsfiddle.net/flexmonster/ojLo9tpw/)
.

8. Configuring report

What is a report?

A report is a definition of what data should be shown in the component and how it will be visualized.

Reports are in JSON format. XML reports are also supported in terms of backward compatibility.

You are already familiar with the report object thanks to Your first pivot table(/doc/your-first-pivot/)
article where the report object was used to visualize pivot table based on JSON data. There were defined such report object properties as data from dataSource, rows, columns and measures from slice. Also, report objects were used to connect to JSON(/doc/json-data-source/)
, CSV(/doc/csv-data-source/)
, SQL databases(/doc/connecting-to-relational-database/)
, Microsoft Analysis Services(/doc/connecting-to-microsoft-analysis-services/)
, Pentaho Mondrian(/doc/connecting-to-pentaho-mondrian/)
and icCube(/doc/connecting-to-iccube/)
in the previous articles of the documentation.

Now it is time to understand what else except the data source configuration can be defined in reports.

Report object has many properties. Actually, all the possible aspects of pivot tables and pivot charts configuration can be set via report object. The component supports saving reports and loading of previously saved ones.

Report parts

Report properties can be divided into the logical parts. Check our tutorials about 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/)

*Only the part that describes data source is the required one in reports. Others are optional.

The simplest report

The simplest report consists only of the data source definition.

Let’s see how the simplest report in JSON format that defines connection to the static CSV file looks like :

```
{
 dataSource: {
  filename: "data.csv"
 }
}
```

This is how the above report is set in $("#pivotContainer").flexmonster():

```
<div id="pivotContainer">The component will appear here</div>
<script type="text/javascript" src="flexmonster/lib/jquery.min.js"></script>
<script type="text/javascript" src="flexmonster/flexmonster.js"></script>

<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  report: {
   dataSource: {
    filename: "data.csv"
   }
  },
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Open on JSFiddle(http://jsfiddle.net/flexmonster/z235ddft/)
.

How to display non-English characters correctly

If you use the 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 also must be encoded as UTF-8.
2. In case you are not able to change the content of your HTML file, you can embed Flexmonster Component in a separate JS file with specified UTF-8 encoding.

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

8.1. Data source

Data source is required part of the report object. Web Pivot Table Component supports the data from OLAP data sources, SQL databases, CSV and JSON static files, inline JSON data. Each data source requires specific properties set inside dataSource section of report object. Read more in the following sections:

- Data from OLAP data sources(#olap)
  - Microsoft Analysis Services(#msas)
  - Mondrian(#mondrian)
  - icCube(#iccube)
- SQL databases(#sql)
- CSV data sources(#csv)
- JSON data sources(#json)
- Change data source using Toolbar(#toolbar)

Data from OLAP data sources

OLAP data sources include MS Analysis Services, Mondrian and icCube. There are two ways to connect to OLAP cube using Flexmonster Pivot Table:

1. via XMLA protocol – an industry standard for data access in analytical systems (for Microsoft Analysis Services, Mondrian and icCube).
2. via our Data Speed Accelerator – special server-side utility developed by Flexmonster (for Microsoft Analysis Services and Mondrian).

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. Given catalog’s cube’s name.
- dataSourceInfo (optional) – String. The service info.
- dataSourceType – String. Type of data source. In this case it is "microsoft analysis services".
- proxyUrl – String. The path to proxy URL.
- binary (optional) – Boolean. Flag to use Data Speed Accelerator instead of 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. Microsoft Locale ID Value for your language.
- roles (optional) – String. Comma-delimited list of predefined roles to connect to a server or database using permissions conveyed by that role. If this property is omitted, all roles are used, and the effective permissions are the combination of all roles. Supported only via XMLA protocol.
- subquery (optional) – String. The parameter to set the server-side filter which helps to decrease the size of the response from the OLAP cube. For example, to show reports only for one specific year set subquery the following way: "subquery": "select {[Delivery Date].[Calendar].[Calendar Year].&[2008]} on columns from [Adventure Works]".

Here is how the connection to MSAS via XMLA represented in dataSource:

```
{
    dataSource: {
        dataSourceType: "microsoft analysis services",
  // URL to msmdpump.dll
        proxyUrl: "http://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 MSAS
        roles: "admin,manager",
        subquery: "select
            {[Delivery Date].[Calendar].[Calendar Year].&[2008]}
            on columns from [Adventure Works]"
    }
}
```

Check out on JSFiddle(http://jsfiddle.net/flexmonster/hcckvwx0/)
.

Here is how the connection to MSAS via Data Speed Accelerator represented in dataSource:

```
{
    dataSource: {
        dataSourceType: "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 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. Given catalog’s cube’s name.
- dataSourceInfo – String. The service info.
- dataSourceType – String. Type of data source. In this case it is "mondrian".
- proxyUrl – String. The path to proxy URL.
- binary (optional) – Boolean. Flag to use Data Speed Accelerator instead of XMLA protocol.
- roles (optional) – String. Comma-delimited list of predefined roles to connect to a server or database using permissions conveyed by that role. If this property is omitted, all roles are used, and the effective permissions are the combination of all roles.

Here is how the connection to Mondrian via XMLA represented in dataSource:

```
{
    dataSource: {
        dataSourceType: "mondrian",
  // URL to XMLA provider
        proxyUrl: "http://olap.flexmonster.com:8080/mondrian/xmla",
        dataSourceInfo: "MondrianFoodMart",
        catalog: "FoodMart",
        cube: "Sales"
    }
}
```

Try the example on JSFiddle(http://jsfiddle.net/flexmonster/uf4tpdz9/)
.

Here is how the connection to Mondrian via Data Speed Accelerator represented in dataSource:

```
{
    dataSource: {
        dataSourceType: "mondrian",
        proxyUrl: "localhost:50006",
        dataSourceInfo: "MondrianFoodMart",
        catalog: "FoodMart",
        cube: "Sales",
        binary: true,
        // Mondrian roles
        roles: "California manager"
    }
}
```

You can read all the details about Accelerator here(/doc/getting-started-with-accelerator-mondrian/)
.

icCube

Here is a list of dataSource properties used to connect to icCube:

- catalog – String. The data source catalog name.
- cube – String. Given catalog's cube's name.
- dataSourceInfo (optional) – String. The service info.
- dataSourceType – String. Type of data source. In this case it is "iccube".
- proxyUrl – String. The path to proxy URL.
- subquery (optional) – String. The parameter to set the server-side filter which helps to decrease the size of the response from the OLAP cube. For example, to show reports only for one specific year set subquery the following way: "subquery": "select {[Delivery Date].[Calendar].[Calendar Year].&[2008]} on columns from [Adventure Works]".

Here is how the connection to icCube via XMLA represented in dataSource:

```
{
    dataSource: {
        dataSourceType: "iccube",
        /* URL to XMLA service */
        proxyUrl: "http://olap.flexmonster.com:8282/icCube/xmla",
        /* Catalog name */
        catalog: "Sales",
        /* Cube name */
        cube: "Sales"
    }
}
```

See it on JSFiddle(http://jsfiddle.net/flexmonster/oz4hqu9t/)
. Read more: Connecting to icCube(/doc/connecting-to-iccube/)
.

SQL databases

Here is a list of dataSource properties used to connect to SQL databases via Compressor:

- dataSourceType – String. Type of data source. In this case it is "ocsv".
- filename – String. The URL to the server-side script which generates data.

Our Flexmonster Data Compressor returns files in OCSV (Optimized CSV) format and you need to define dataSourceType explicitly:

```
{
    dataSource: {
        dataSourceType: "ocsv",
        /* URL to the Data Compressor .Net, Java or PHP */
        filename: "http://localhost:55772/api/flexmonster/get"
    }
}
```

Read here(/doc/connecting-to-relational-database/)
more about our Data Compressor.

CSV data sources

CSV data source can be:

- file from the local file system
- remote static file
- data generated by the 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 the file from the local file system (true) or not (false). Default value is false.
- dataSourceType – String. Type of data source. In this case it is "csv".
- fieldSeparator – String. Defines specific fields separator to split CSV row. There is no need to define it if CSV fields are separated by , or ;. This property is used if another char separates fields. For example, if you use TSV, where tab char is used to separate fields in the row, fieldSeparator parameter should be defined as "\t".
- filename – String. The URL to the file or to the server-side script which generates data.
- ignoreQuotedLineBreaks (starting from v2.1) – Boolean. Indicates whether the line breaks in quotes will be ignored (true) in CSV files or not (false). Default value is true, which makes CSV parsing faster. Set it to false only if your data source has valuable for you line breaks in quotes. Please note that this might slow down CSV parsing a little bit.
- recordsetDelimiter – String. Defines which char is used in CSV to denote the end of CSV row. Default value is "?".

In the following example data is taken from CSV file where colon char (:) is used to separate fields in the row. Line breaks in quotes are not ignored:

```
{
    dataSource: {
        /* The URL to CSV file or local path */
        filename: 'colon-data.csv',
        fieldSeparator: ':',
        ignoreQuotedLineBreaks: false,
    }
}
```

If data is generated by the server-side script, dataSourceType should be defined explicitly:

```
{
    dataSource: {
        dataSourceType: "csv",
        filename: "script_which_returns_csv_file"
    }
}
```

JSON data sources

JSON data source can be:

- file from the local file system
- data generated by the 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 is false.
- data (starting from v2.2) – Property to set JSON data if it is already on the page.
- dataSourceType – String. Type of data source. In this case it is "json".
- filename – String. The URL to the file or to the server-side script which generates data.

JSON data set through the file from the local file system:

```
{
    dataSource: {
        /* Path to the local JSON file */
        filename: "data.json"
    }
}
```

If data is generated by the server-side script, dataSourceType should be defined explicitly:

```
{
    dataSource: {
        dataSourceType: "json",
        filename: "script_which_returns_json_file"
    }
}
```

Inline JSON:

```
{
    dataSource: {
        /* jsonData variable contains JSON data */
        data: jsonData
    }
}
```

Change data source using Toolbar

Please use Connect to choose another data source or Open to load another report in run time. Use Save to save the report with the current data source.

Data in pivot table will be updated and will be saved within the report.

Data source via API

API calls connectTo()(/api/connectto/)
, load()(/api/load/)
and open()(/api/open/)
are used to change data source in run time. API call save()(/api/save/)
is used to save the report.

8.2. Slice

Slice is a definition of what data subset from your data source is going to be shown in your report when you open it. By using slice in a report you can easily switch between different sets of values. Find more details below in the 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 slice using the Fields List and controls on the grid(#change)

Slice properties

You can define fields that go to rows, go to columns and go to measures, add filtering, sorting, report filtering, expands and drills. Here is a list of all available properties for 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, hierarchy unique name.
    - caption (optional) - String, hierarchy caption;
    - dimensionName (optional) - String, dimension name;
    - filter (optional) - object with the information on filtering:
        - members - Array of hierarchy's members to be reflected/shown according to the applied filter;
        - negation - Boolean. Represents the filter on hierarchy's members. It tells the component to show the members of hierarchy specified in items (true) or to show all the members of hierarchy except the items (false).
        - measure - Represents the filter on values. The name of the measure on which Top X or Bottom X filter will be applied.
        - quantity - Represents the filter on values. Number of elements to choose for the Top X filter if type is 'top' or for the Bottom X filter if type is 'bottom'.
        - type - Represents the filter type applied to the hierarchy. It can be: 'none' - filter is not applied to the hierarchy, 'members' - the filter on hierarchy's members is applied, 'top' - the filter Top X is applied on values, 'bottom' - the filter Bottom X is applied on values.
    - levelName (optional) - String. Used to specify the level of the hierarchy that is shown on the grid.
    - sort (optional) - String, sorting type for the members ("asc", "desc" or "unsorted");
    - sortOrder (optional) - Array. Using this property you can set custom sorting for hierarchy members. You can specify sortOrder the following way: [“member_1”, “member_2”, etc.].
- drills (optional) - Object. Stores the information about drill-downs in multilevel hierarchies.
    - drillAll (optional) - Boolean. Indicates whether all levels of all hierarchies in slice will be drilled down (true) or drilled up (false).
    - columns (optional) - Array of objects. It is used to save and restore drilled down columns.
    - rows (optional) - Array of objects. It is used to save and restore drilled down rows.
- expands (optional) - Object. Stores the information about expanded nodes.
    - expandAll (optional) - Boolean. Indicates whether all nodes in the data tree will be expanded (true) or collapsed (false) on the grid and on charts.
    - columns (optional) - Array of objects. It is used to save and restore expanded columns.
    - rows - Array of objects. It is used to save and restore expanded rows.
- measures - Array of objects. A list of selected measures and those which have different properties from default ones. Each object has the following properties:
    - uniqueName - String, measure unique name.
    - active (optional) - Boolean value that defines whether the measure will be in the list of available values but not selected (false) or will be selected for the report (true).
    - aggregation (optional) — String, unique name of aggregation that will be applied to the measure ("sum", "count", "distinctcount", "average", "product", "min", "max", "percent", "percentofcolumn", "percentofrow", "index", "difference", "%difference"). If it is calculated measure, it will be "none".
    - availableAggregations (optional) — Array of strings that represents the list of aggregation functions which can be applied to the current measure. If it is calculated measure, it will be [].
    - caption (optional) - String, measure caption.
    - formula (optional) - String that represents the formula that can contain the following operations: +, -, *, /; other measures can be addressed using measure unique name and aggregation function, for example sum("Price") or max("Order"). Pivot supports the following aggregation functions for CSV, OCSV and JSON data sources: "sum", "count", "distinctcount", "average", "product", "min", "max", "percent", "percentofcolumn", "percentofrow", "index", "difference", "%difference".
    - individual (optional) - Boolean. Defines whether the formula is calculated using raw values (true) or using aggregated values (false). Default value is false.

    - format (optional) - String, name of number formatting.
    - grandTotalCaption (optional) - String, measure grand total caption.
- memberProperties - Array of objects. Each object has the following properties:
    - levelName – String, hierarchy unique name.
    - properties - Array of member properties, which will be shown on the grid. Other available member properties can be accessed through the context menu.
- pages - Array of objects. A list of hierarchies selected in the report slice for pages (“Report Filter”). Each object has the following properties:
    - uniqueName - String. Hierarchy unique name.
    - caption (optional) - String, hierarchy caption;
    - dimensionName (optional) - String, dimension name;
    - filter (optional) - object with the information on filtering:
        - members - Array of hierarchy's members to be reflected/shown according to the applied filter;
        - negation - Boolean. It tells the component to show the members of hierarchy specified in items (true) or to show all the members of hierarchy except the items (false).
    - levelName (optional) - String. Used to specify the level of the hierarchy that is shown on the grid.
    - sort (optional) - String. Sorting type for the members ("asc", "desc" or "unsorted");
    - sortOrder (optional) - Array. Using this property you can set custom sorting for hierarchy members. You can specify sortOrder the following way: [“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, hierarchy unique name.
    - caption (optional) - String, hierarchy caption;
    - dimensionName (optional) - String, dimension name;
    - filter (optional) - object with the information on filtering:
        - members - Array of hierarchy's members to be reflected/shown according to the applied filter;
        - negation - Boolean. Represents the filter on hierarchy's members. It tells the component to show the members of hierarchy specified in items (true) or to show all the members of hierarchy except the items (false).
        - measure - Represents the filter on values. The name of the measure on which Top X or Bottom X filter will be applied.
        - quantity - Represents the filter on values. Number of elements to choose for the Top X filter if type is 'top' or for the Bottom X filter if type is 'bottom'.
        - type - Represents the filter type applied to the hierarchy. It can be: 'none' - filter is not applied to the hierarchy, 'members' - the filter on hierarchy's members is applied, 'top' - the filter Top X is applied on values, 'bottom' - the filter Bottom X is applied on values.
    - levelName (optional) - String. Used to specify the level of the hierarchy that is shown on the grid.
    - sort (optional) - String, sorting type for the members ("asc", "desc" or "unsorted");
    - sortOrder (optional) - Array. Using this property you can set custom sorting for hierarchy members. You can specify sortOrder the following way: [“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. Object has 3 properties:
        - tuple - Array. Consists of unique names that identifies the column in the table based on data in it;
        - measure – String. Measure unique name on which sorting will be applied;
        - type - String, sorting type ("asc" or "desc").
    - row - Object. Defines the sorting for numbers in a specific row. Object has 3 properties:
        - tuple - Array. Consists of unique names that identifies the row in the table based on data in it;
        - measure – String. Measure unique name on which sorting will be applied;
        - type - String, sorting type ("asc" or "desc").

Default slice

If slice was not defined, Flexmonster will select a default slice for the report, where the first hierarchy from data goes to rows and the first measure goes to columns. The automatic default slice selection available for JSON, CSV and OCSV data sources (it is not available for OLAP). You can turn off default slice by setting showDefaultSlice from options to false. For example, have a look at 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, default slice will look the following way:

```
{
    "dataSource": {
        "data": jsonData
    },
    "slice": {
        "rows": [
            {
                "uniqueName": "Category"
            }
        ],
        "columns": [
            {
                "uniqueName": "[Measures]"
            }
        ],
        "measures": [
            {
                "uniqueName": "Price"
            }
        ]
    }
}
```

See the same example with default slice on JSFiddle(https://jsfiddle.net/flexmonster/22ospzoa/)
.

Rows, columns and measures

Starting from version 2.304, slice can be defined just with measures:

```
{
    "dataSource": {
        "filename": "http://www.flexmonster.com/download/data.csv"
    },
    "slice": {
        "measures": [
            {
    "uniqueName": "Price",
    "aggregation": "sum",
    "active": true
            }
        ]
    }
}
```

Open on JSFiddle(http://jsfiddle.net/flexmonster/xnfyLff0/)
.

"uniqueName": "[Measures]" allows to define where measures will be displayed (in rows or in columns). By default they go to columns. This is the example of slice with rows, columns and measures:

```
{
    "dataSource": {
        "filename": "http://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",
                "availableAggregations": ["min", "max"],
                "active": true
            }
        ]
    }
}
```

See the same example on JSFiddle(http://jsfiddle.net/flexmonster/a2mguf82/)
.

Report filter

Report filter allows displaying different data subsets in the report. In Pivot Component pages property is used for defining report filters as follows:

```
{
    "dataSource": {
        "filename": "http://www.flexmonster.com/download/data.csv"
    },
    "slice": {
        "pages": [
            {
                "uniqueName": "Color",
                "filter": {
                    "members": [
                        "color.[yellow]",
                        "color.[white]"
                    ]
                }
            }
        ],
        "rows": [
            {
                "uniqueName": "Category"
            }
        ],
        "columns": [
            {
                "uniqueName": "[Measures]"
            }
        ],
        "measures": [
            {
                "uniqueName": "Price"
            }
        ]
    }
}
```

See the example with report filter on JSFiddle(http://jsfiddle.net/flexmonster/7jpbqjsh/)
.

Sorting

This 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 the following way:

```
"sorting": {
    "column": {
        "type": "desc",
        "tuple": [],
        "measure": "Price"
    }
}
```

Sorting for the members of columns and rows is defined like this:

```
"rows": [
    {
        "uniqueName": "Category",
        "filter": {
            "members": [
                "category.[cars]",
                "category.[bikes]"
            ]
        },
        "sort": "desc"
    }
]
```

Expands

The information about expands and collapses is stored within the slice. When the customer performs one of these operations, all the changes can be saved within the report. Use expandAll property to apply the operation to all levels of data detail at once. This is how expands object looks:

```
"expands": {
    "expandAll": false,
    "rows": [
        {
            "tuple": [
                "category.[accessories]"
            ]
        },
        {
            "tuple": [
                "category.[cars]"
            ]
        }
    ]
}
```

Drills

drills object is used to store the information about drill-downs in multilevel hierarchies. Here is the example of drills object:

```
"drills": {
    "drillAll": false,
    "rows": [
        {
            "tuple": [
                "category.[accessories]"
            ]
        },
        {
            "tuple": [
                "category.[accessories].[bike racks]"
            ]
        },
        {
            "tuple": [
                "category.[accessories].[bottles and cages]"
            ]
        }
    ]
}
```

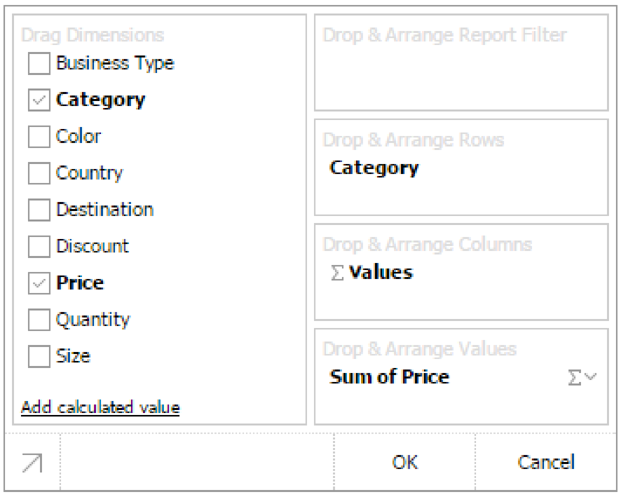
Calculated values

You can create as many calculated measures for one report as you need, there is no limit towards the number of calculated measures. When you save the report all the calculated measures will be saved as well and loaded when the report is retrieved. Please note that you can add calculated measures only for reports based on JSON, CSV, OCSV data source. Below is the example of such calculated measure:

```
"measures": [
    {
        "uniqueName": "Avg",
        "formula": "sum('Price') / count('Category') ",
        "caption": "Avg",
        "active": true
    }
]
```

Change slice using the Fields List and controls on the grid

Please use the Fields List to define report filters, rows, columns and values in run time.



Sorting, filtering, expand/collapse operations, drill up and down are available directly on the grid and on built-in pivot charts.

Slice via API

You can change the slice among with other report parts using API call setReport()(/api/setreport/)
. To change only the slice please use runQuery()(/api/runquery/)
call.

8.3. Options

The way how Pivot Component looks can be defined within the report. Options are used to specify appearance and functionality available for customers. Default options are set for every pivot instance. Specific options can be defined in addition to default ones. Find more details in the following sections:

- Options properties(#properties)
- Default options(#default)

- Grid options(#grid)
- Chart options(#chart)
- Change options using Toolbar(#toolbar)

Options properties

With options you can define Fields List mode, enable/disable drill through, showing of default slice, sorting, editing, define grid and chart properties. Here is a list of all available properties for options:

- viewType - String. Type of view to show: "grid" or "charts" or "grid_charts" (starting from v1.9).
- grid – Object. Contains information about grid:
  - type - String. Selected grid type. The following grid types are supported: "compact", "classic" and "flat".
  - title - String. Title of the grid.
  - showFilter - Boolean. Indicates whether the opening columns/rows filter controls and page filter controls are visible (true) or not (false) on the grid. Default value is true.
  - showHeaders - Boolean. Indicates whether the spreadsheet headers are visible (true) or not (false).
  - fitGridlines - Boolean. Indicates whether the gridlines are shown for all cells (false) or only non-empty (true).
  - showTotals - Boolean. Indicates whether the totals are visible (true) or not (false).
  - showGrandTotals - String. Specifies how to show grand totals: in rows ("rows"), in columns ("columns"), in rows and columns ("on") or hide them ("off"). Default value is "on".
  - showHierarchies - Boolean. Specifies how to show drillable hierarchy cells on the grid: with link on the right (true) or with icon on the left (false). Default value is true.
  - showHierarchyCaptions - Boolean. Indicates whether the hierarchy captions are visible (true) or not (false) on the grid. Default value is true.
  - showReportFiltersArea (starting from v2.2) - Boolean. Indicates whether the reports filtering cells on the grid should be visible (true) or not (false). Default value is true.
  - pagesFilterLayout (starting from v2.3) - String. Allows to choose the layout for the report filters. Possible values: "horizontal" and "vertical". Default value is "horizontal".
  - drillthroughMaxRows (starting from v2.318) - Number. Allows setting the maximal number of rows for the MSAS Drill Through popup. Supported only via XMLA protocol. The default value is 1000.
- chart – Object. Contains information about charts:
  - type - String. Selected chart type. The following chart types are supported: "bar", "bar_h" (Horizontal Bar), "line", "scatter", "pie", "bar_stack" and "bar_line" (starting from v1.9).
  - title - String. Title of the chart.
  - showFilter (starting from v2.2) - Boolean. Indicates whether the opening columns and rows filter controls are visible (true) or not (false) on the charts. Default value is true.
  - multipleMeasures - Boolean. Starting from v1.9. Indicates whether to show multiple measures on charts. Default value is false.
  - oneLevel - Boolean. In a case of 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 on the legend (true). Default value is false.
  - autoRange - Boolean. Indicates whether the range of values on the charts is selected automatically or not.
  - showLegendButton (starting from v2.2) - Boolean. Indicates whether the button to show/hide the legend on charts is visible. Default value is false which means that the legend is always visible, without the button that hides it.
  - showAllLabels - Boolean. Setting a value to true allows showing all the labels in Pie ?hart. If the value is false it will have the same behavior as it was before. Default value is false.

  - showMeasures (starting from v2.2) - Boolean. Hides all dropdowns on the top of charts if you want to show simple chart without controls or you want to save space. Default value is true - the dropdowns are visible by default, as it was in previous versions.
  - showOneMeasureSelection - Boolean. The default value is true, which means that the visibility of the measures dropdown on charts does not depend on the amount 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.
  - showWarning - Boolean. Indicates whether the warning are shown if data is too big for charts.
  - position - String. Position of charts related to the grid. It can be "bottom", "top", "left" or "right". Default value is "bottom".
  - activeMeasure - String. Selected measure on charts view.
  - pieDataIndex - String. Selected tuple index on the Pie chart.
- configuratorActive - Boolean. Indicates whether the Fields List is opened (true) or closed (false).
- configuratorButton - Boolean. Indicates whether the Fields List toggle button is visible (true) or not (false).
- configuratorMatchHeight (starting from v2.1) - Boolean. Indicates whether the Fields List will be the same height as the component (true) or it's height will be defined by its content amount (false). Default value is false.
- showAggregations (starting from v2.0) - Boolean. Indicates whether the aggregation selection control is visible (true) or not (false) for measures on Fields List. Default value is true.
- showCalculatedValuesButton (starting from v2.2) - Boolean. Controls the visibility of "Add calculated value" in Fields List. Default value is true.
- grouping - Boolean. Indicates whether grouping is enabled. Default value is false. This feature allows customers to group chosen elements using filter window. For example, the customer has shops in different cities and wants to analyze sales information. It is possible to combine several cities in one group by geography or by the sales numbers, etc. Only for "ocsv", "csv" and "json" data source types.
- editing (starting from v2.1) - Boolean. Indicates whether the editing feature is enabled (true) or disabled (false) on the Drill Through popup for CSV, OCSV and JSON data sources. User will be able to double-click the cell and enter new value in it if the editing feature is enabled.
- drillThrough (starting from v2.1) - Boolean. Indicates whether the drill through feature is enabled (true) or disabled (false). User can drill through by double-clicking the cell with value. Drill through feature is available for all data sources except icCube. Default value is true.
- showDrillThroughConfigurator - Boolean. Indicates whether the Fields List toggle button is visible in Drill Through view. Default value is true.
- sorting (starting from v2.0) - 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 ("off" or false). Default value is "on".
- datePattern - String. It is used to format "date string" date fields ("type":"date string" in JSON, "ds+" prefix in CSV). A default pattern string is dd/MM/yyyy.
- dateTimePattern - String. It is used to format "datetime" date fields ("type":"datetime" in JSON, "dt+" prefix in CSV). A default pattern string is dd/MM/yyyy HH:mm:ss.
- saveAllFormats - Boolean. If there are more than 5 formats defined, only the formats that are used for "active=true" measures will be saved in the report. In order to get saved all the formats, no matter how many of them you have and whether they are used for active measures or not, please set saveAllFormats property to true. Default value is false.
- showDefaultSlice (starting from v2.2) - Boolean. Defines whether the component selects a default slice for the report with empty slice (when nothing is set in rows, columns, pages 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, please set this property to false. Default value is true. Only for "csv", "ocsv" and "json" data source types.
- useOlapFormatting (optional) - 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 is false.
- showMemberProperties - Boolean. Indicates whether the member properties for OLAP data source are available in the component (true) or not (false). Default value is false. This feature is only for "microsoft analysis services" and "mondrian" data source types.
- showEmptyData - Boolean. By default, if you have an empty CSV data source but the header is defined

the component will show your slice with empty data cells. Set the value as false - the Component will show the “Data source is empty. Please check the CSV file.” message.
- defaultHierarchySortName (starting from v2.0) - String. Sorting type for hierarchies' members ("asc", "desc" or "unsorted"). Default value is "asc".
- selectEmptyCells (starting from v2.3) - Boolean. Indicates whether it is possible to select cells outside of the table. Default value is true.
- showOutdatedDataAlert - Boolean. Setting a value to true will show the warning to the user before automatic reloading of data from the cube. Default value is false which means there will be no warnings. Only for Flexmonster Accelerator.
- showAggregationLabels - Boolean. Indicates whether aggregation labels like "Total Sum of", "Sum of", etc. are shown in the columns/rows title. Default value is true.

Default options

In case options were not defined in the report, Pivot Component will use global options, if they are defined, or defaults from the component. These options can be overridden in the report. Below is an example of default options. Please note, you have to specify only necessary options. The rest will be defined automatically by the component instance.

```
{
   "dataSource": {
        "filename": "http://www.flexmonster.com/download/data.csv"
    },
    "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
    }
}
```

Grid options

All grid options are combined in grid section of options object. Here you can specify just the necessary properties. This example shows how to set title and hide totals from the grid:

```
{
    "dataSource": {
        "filename": "http://www.flexmonster.com/download/data.csv"
    },
    "options": {
        "grid": {
            "title": "Results",
            "showTotals": false
        }
    }
}
```

Check out on JSFiddle(http://jsfiddle.net/flexmonster/2rn0bxgo/)
.

Chart options

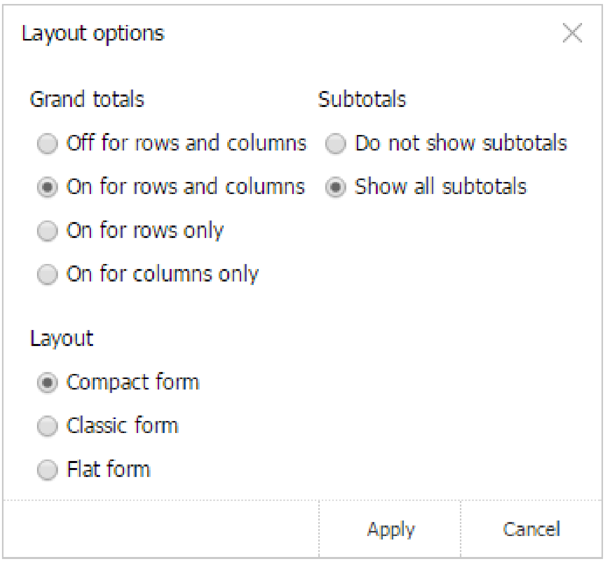
All chart options are combined in chart section of options object. The following example will set chart title, type and hide measure dropdowns on the top of charts. Please note, viewType is set to charts to see the result instantly:

```
{
    "dataSource": {
        "filename": "http://www.flexmonster.com/download/data.csv"
    },
    "options": {
        "viewType": "charts",
        "chart": {
            "type": "scatter",
            "title": "Summary chart",
            "showMeasures": false
        }
    }
}
```

Open the example on JSFiddle(http://jsfiddle.net/flexmonster/bmvhd7bt/)
.

Change options using Toolbar

Please use Options in Toolbar to change grand totals, subtotals and table layout in run time.


Layout options

Grand totals
- Off for rows and columns
- [✓ On for rows and columns]
- On for rows only
- On for columns only

Subtotals
- Do not show subtotals
- [✓ Show all subtotals]

Layout
- [✓ Compact form]
- Classic form
- Flat form

Options via API

You can change options among with other report parts using API call setReport()(/api/setreport/)
. Also options can be changed separately via setOptions()(/api/setoptions/)
. To see the current options use getOptions()(/api/getoptions/)
.

8.4. Number formatting

The way how numeric values are formatted in the component can be defined in a report.

A default format is for all measures. The specific number formats can be defined for some measures in addition to the default format. More details are below in the following sections:

- Number format properties(/doc/number-formatting/#properties)

- Default number format(/doc/number-formatting/#default)

- Number format for a specific measure(/doc/number-formatting/#currency)

- Change number formatting using Toolbar(/doc/number-formatting/#toolbar)

If the component is connected to OLAP cube and you already have formatted numbers there (Microsoft Analysis Services or Mondrian), you can display these formatted values without applying number formatting in the component. See more about this option below:

- Number formatting from OLAP cube(/doc/number-formatting/#fromcube)

Number format properties

With number formatting you can define thousands and decimal separators, the number of decimals to show in the fractional part of a number, how to display null and infinity values, you can format number as currencies specifying the currency symbol and its position – right or left.

Here is a list of all available properties:

- name – String. It identifies the format in the report, thus, it should be unique. The default is "", which means that this number format is a default one and it is applied to all the measures for which the specific number format is not set.
- thousandsSeparator – String. The default is " " (space).
- decimalSeparator – String. The default is ".".
- decimalPlaces – Number. The exact number of decimals to show in the fractional part of a number after the decimal separator. The default is -1, which means that the number will be shown as is.
- maxDecimalPlaces – Number. The maximum number of decimals to show in the fractional part of a number after the decimal separator. The default is -1, which means the number will be shown as is.
- maxSymbols – Number. The maximum number of symbols in a cell. The default is 20.
- currencySymbol – String. The symbol which is shown near the value (currency symbol, hours, percent, etc.). The default is "".
- currencySymbolAlign – String. The alignment of the currency symbol. It can be "left" or "right". The default is "left".
- isPercent – Boolean. It allows to format data as percentage. The behavior is the same as in Excel. The default is false. Set isPercent to true and numbers will be multiplied by 100 and % symbol will be added. For example, 0.56 will be changed to 56%. Please note, if % is set as currencySymbol, setting isPercent to true will not multiply numbers by 100.
- nullValue – String. It defines how to show null values in the grid. The default is "".
- infinityValue – String. It defines how to show infinity values in the grid. The default is "Infinity".
- divideByZeroValue – String. It defines how to show divided by zero values in the grid. The default is "Infinity".
- textAlign – String. The alignment of formatted values in cells on the grid: "right" or "left". The default is "right".

Default number format

Flexmonster component has a built-in default number format which 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.

If you want to override the default number format for a report, please 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,
   currencySymbol: "",
   currencySymbolAlign: "left",
   isPercent: "false",
   nullValue: "",
   infinityValue: "Infinity",
   divideByZeroValue: "Infinity",
   textAlign: "right"
  }
 ],
 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(http://jsfiddle.net/flexmonster/xjav1g36/)
.

Number format for a specific measure

A number format can be applied to a specific measure or measures. Each measure has only one format but a format can be applied to more than one measure.

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 the format to the specific measure, two things should be done:

- a format should be named,
- the format name should be defined for the measure(-s) in a default slice.

When you define some properties in the default format they will be applied to all other formats. In the following example each measure with number formats currency and amount will have thousandsSeparator: ”,”, because it was defined in the default format:

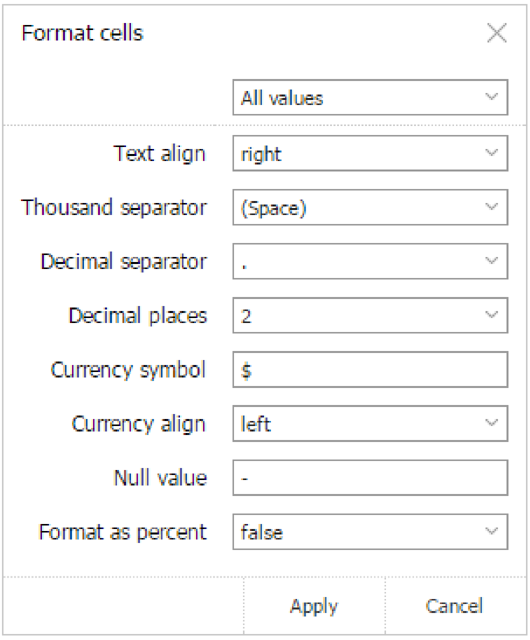
```
{
 dataSource: {
  filename: "http://www.flexmonster.com/download/data.csv"
 },
 formats: [
  {
   name: "",
   thousandsSeparator: ","
  },
  {
   name: "currency",
   currencySymbol: "$"
  },
  {
   name: "amount",
   decimalPlaces: 0,
   currencySymbol: " pcs.",
   currencySymbolAlign: "right"
  }
 ],
 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(http://jsfiddle.net/flexmonster/qxyse7n6/)
.

Please note that a format can be defined for the measure(-s) even if they are not active (active property is false) in a default slice.

Change number formatting using Toolbar

Please use Format > Format cells in Toolbar to change/define number formatting for measures in run time.



	All values
Text align	right
Thousand separator	(Space)
Decimal separator	.
Decimal places	2
Currency symbol	$
Currency align	left
Null value	-
Format as percent	false

The number format will be applied to the measures and will be saved within the report.

Number formatting via API

API calls setFormat()(/api/setformat/)
and getFormat()(/api/getformat/)
are used to manipulate number formatting in run time.

Number formatting from OLAP cube

If you already have formats defined for measures in OLAP cube and you want to use the formatted values from the cube, please set useOlapFormatting report property to true to enable this (it is turned off by default), as follows:

```
{
 dataSource: {
  dataSourceType: "microsoft analysis services",
  proxyUrl: "http://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(http://jsfiddle.net/flexmonster/npu21mke/)
.

Please note that useOlapFormatting is supported for Microsoft Analysis Services via both Accelerator and XMLA, and for Mondrian via Accelerator. It is not available for Mondrian via XMLA and for icCube.

8.5. Conditional formatting

Conditional formatting allows highlighting cells depending on their values. You can create as many conditions with different formatting for one report as you need, there is no limit towards the number of conditions. If there are more than one conditional formatting rules for the report, they will be applied one by one in the order they have been created.

The conditional formatting 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 selected measure, hierarchy, and hierarchy’s member.

Conditions can be defined within a report. When you save the report all the conditional formatting will be saved as well and loaded when the report is retrieved.

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 Toolbar(#toolbar)

Conditional format properties

With conditional formatting, you can define a logical expression for condition rules, style objects for matched and mismatched cells, and application range for conditions. 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 - IF statement with 3 arguments: IF(CONDITION, TRUE STYLE, FALSE STYLE), where the false style is optional. Condition can contain AND, OR, ==, !=, >, <, >=, <=, +, -, *, /, isNaN(), !isNaN(). #value is used to address the cell value in condition. Example: if(#value > 2, "trueStyle", "falseStyle").
- trueStyle - style object that will be applied to a cell if the condition for the cell value is met. Please note that for export to Excel and PDF colors should be set as hex color codes.
- falseStyle (optional) - style object. If it is set it will be applied to a cell if the condition for the cell value is not met.
- id (optional) - id of the conditional formatting rule. If id property is not set, the id for the rule will be set inside Pivot component.
- row (optional) - row index to which the condition should be applied.
- column (optional) - column index to which the condition should be applied.
- measure (optional) - measure unique name to which the condition should be applied.
- hierarchy (optional) - hierarchy unique name to which the condition should be applied.
- member (optional) - member unique name to which the condition should be applied.
- isTotal (optional) - Boolean. If it is not defined, the condition will be applied to all cells. If it is true, the condition will be applied to totals and subtotals cells only. If it is false, the condition will be applied to regular cells only.

Style object format

trueStyle and falseStyle are both style objects, but the latter is optional. Style object can have the following properties:

```
"trueStyle": {
    "backgroundColor": "#FFFFFF",
    "color": "#0000FF",
    "fontFamily": "Arial",
    "fontSize": 12
}
```

You can specify only necessary properties.

In case you want to export the pivot table to Excel and PDF don’t forget to set colors as hex color codes.

Conditional formatting for all values

You need to specify properties formula and trueStyle to apply the conditional rule to all values. You can define format the following way:

```
{
    "dataSource": {
        "filename": "http://www.flexmonster.com/download/data.csv"
    },
    "conditions": [
        {
            "formula": "if(#value < 400000, 'trueStyle')",
            "trueStyle": {
                "backgroundColor": "#FFFFFF",
                "color": "#0000FF",
                "fontFamily": "Arial",
                "fontSize": 12
            }
        }
    ]
}
```

Open the example on JSFiddle(http://jsfiddle.net/flexmonster/zypcc8tx/)
.

Conditional formatting for specific values

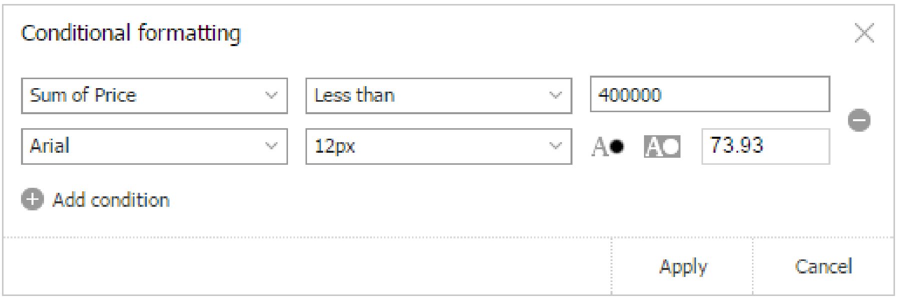
The formatting rule can be applied to a specific measure, hierarchy, hierarchy member, column or row. Also, you can apply it only to regular cells or totals and subtotals. For example, if you are visualizing financial data, you may want to apply conditional formatting only to regular cells with prices. Please see an example below:

```
{
    "dataSource": {
        "filename": "http://www.flexmonster.com/download/data.csv"
    },
    "conditions": [
        {
            "formula": "if(#value < 400000, 'trueStyle')",
            "measure": "Price",
            "isTotal": false,
            "trueStyle": {
                "backgroundColor": "#FFFFFF",
                "color": "#0000FF",
                "fontFamily": "Arial",
                "fontSize": 12
            }
        }
    ],
    "slice": {
        "rows": [ {"uniqueName": "Category"} ],
        "measures": [
            {"uniqueName": "Price"},
            {"uniqueName": "Quantity"}
        ]
    }
}
```

Open the example on JSFiddle(http://jsfiddle.net/flexmonster/gd7woe0g/)
.

Change conditional formatting using Toolbar

Please use Format > Conditional formatting in Toolbar to change/define conditional formatting for values in run time.



This conditional formatting will be applied to the specified values and will be saved within the report.

?onditional formatting via API

API call addCondition()(/api/addcondition/)
is used to add or change the conditional formatting rule in run time. You can change conditions among with other report parts using API call setReport()(/api/setreport/)
.

8.6. Set report to the component

There are several ways to set report in JSON or XML format to the component:

- Using report object in $(“#pivotContainer”).flexmonster() API call

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

<script>
 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 = $("#pivotContainer").flexmonster({
  toolbar: true,
  report: {
   dataSource: {
    data: jsonData
   },
   slice: {
    rows: [
    { uniqueName: "Color" },
    { uniqueName: "[Measures]" }
    ],
    columns: [
    { uniqueName: "Country" }
    ],
    measures: [
    {
     uniqueName: "Price",
     aggregation: "sum"
    }
    ]
   }
  },
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });
</script>
```

Open the example on JSFiddle(http://jsfiddle.net/flexmonster/5bv9bfbr/)
.

- Using setReport() API call

```
<button onclick="setReport()">Set report</button>
<div id="pivotContainer">The component will appear here</div>
<script src="flexmonster/lib/jquery.min.js"></script>
<script src="flexmonster/flexmonster.js"></script>

<script>
 var pivot = $("#pivotContainer").flexmonster({
  toolbar: true,
  licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
 });

 function setReport() {
  var report = flexmonster.getReport();
  // parse, change or save for later use
  flexmonster.setReport(report);
 }
</script>
```

Try on JSFiddle(http://jsfiddle.net/flexmonster/u981gsL2/)
.

- Using open() API call to select JSON file from the local file system. In previous versions, reports were in XML format. You can open them as well as JSON.

  ```
  <button onclick="openReport()">Open report JSON</button>
  <div id="pivotContainer">The component will appear here</div>
  <script src="flexmonster/lib/jquery.min.js"></script>
  <script src="flexmonster/flexmonster.js"></script>

  <script>
   var pivot = $("#pivotContainer").flexmonster({
    toolbar: true,
    licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
   });

   function openReport() {
    flexmonster.open();
   }
  </script>
  ```

  Check out on JSFiddle(http://jsfiddle.net/flexmonster/pzt3ynt1/)
  .

- Using load() API call to load JSON file from URL. In previous versions, reports were in XML format. You can open them as well as JSON.

  ```
  <button onclick="loadReport()">Load report JSON</button>
  <div id="pivotContainer">The component will appear here</div>
  <script src="flexmonster/lib/jquery.min.js"></script>
  <script src="flexmonster/flexmonster.js"></script>

  <script>
   var pivot = $("#pivotContainer").flexmonster({
    toolbar: true,
    licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
   });

   function loadReport() {
    flexmonster.load("report.json");
   }
  </script>
  ```

  Try on JSFiddle(http://jsfiddle.net/flexmonster/t78pfd0o/)
  .

8.7. Get report from the component

There are several ways to get report in JSON format from the component:

- You can get report from the component using getReport() API call

  ```
  <button onclick="getReport()">Get report</button>
  <div id="pivotContainer">The component will appear here</div>
  <script src="flexmonster/lib/jquery.min.js"></script>
  <script src="flexmonster/flexmonster.js"></script>

  <script>
   var pivot = $("#pivotContainer").flexmonster({
    toolbar: true,
    licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
   });

   function getReport() {
    var report = flexmonster.getReport();
    // parse, change or save for later use
    flexmonster.setReport(report);
   }
  </script>
  ```

  Open the example on JSFiddle(http://jsfiddle.net/flexmonster/8hrmnc9L/)
  .

- Using save() API call to save JSON report to the server or to the local file system

  ```
  <button onclick="saveReport()">Save report JSON</button>
  <div id="pivotContainer">The component will appear here</div>
  <script src="flexmonster/lib/jquery.min.js"></script>
  <script src="flexmonster/flexmonster.js"></script>

  <script>
   var pivot = $("#pivotContainer").flexmonster({
    toolbar: true,
    licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
   });

   function saveReport() {
    flexmonster.save("report.json", "file");
   }
  </script>
  ```

  Open the example on JSFiddle(http://jsfiddle.net/flexmonster/a21uj95v/)
  .

The easiest way to create report JSON file is to open Pivot Table demo(/demos)
that is already connected to the sample CSV data and click Save button on the Toolbar.

Here is a sample of such a report file:

```
{
    "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"
}
```

8.8. Date and time formatting

This article explains how to define a format for date and time strings representation inside the component.

Input date format

As an input date format, pivot table component supports ISO 8601(http://www.w3.org/TR/NOTE-datetime) date (other formats may be used, but results can be unexpected). For example, "2016-03-20" (just date) or "2016-03-20T14:48:00" (date and time).

Representation format

The format for representation of dates and time inside the component differs from the input format. The component supports the pattern strings to format date and time data. There are two properties of a report to format date and time strings:

1. datePattern. It is used to format “date string” date fields ("type":"date string" in JSON, "ds+" prefix in CSV). A default pattern string is dd/MM/yyyy.
2. dateTimePattern. It is used to format “datetime” date fields ("type":"datetime" in JSON, "dt+" prefix in CSV). A default pattern string is dd/MM/yyyy HH:mm:ss.

In order to change the default date and time format for “date string” date fields, you need to specify datePattern in the report explicitly. The same is for “datetime” date fields format, so you need to specify dateTimePattern in the report.

Here is how to define datePattern in the report:

```
{
 dataSource: {
  dataSourceType: "json",
  data: [
  {
   "date":{"type":"date string"},
   "n":{"type":"number"}
  },
  {
   "date":"2016-04-06T23:59:30",
   "n":1
  },
  {
   "date":"2016-04-06T23:59:30",
   "n":1
  },
  {
   "date":"2016-02-07T20:33",
   "n":1
  }
  ]
 },
 options: {
  datePattern: "yyyy-MM-dd HH:mm:ss"
 },
 slice: {
  rows: [{ uniqueName: "date" }],
  columns: [{ uniqueName: "[Measures]" }],
  measures: [{ uniqueName: "n" }],
 }
}
```

Test or modify this example here(http://jsfiddle.net/flexmonster/tc8yosgd/)
.

The same can be done for dateTimePattern in the report. Please note that both
patterns datePattern and dateTimePattern can contain time part.

Also, user’s local time zone is used to represent the date by default. If it is required to use UTC time zone, please include UTC: part at the beggining of the pattern (i.e. **UTC:**dd/MM/yyyy HH:mm:ss).

Patterns syntax

The pattern contains sequences of letters that are replaced with date and time values in the formatted string. For example, in the pattern "yyyy/MM" the characters "yyyy" are replaced with a four-digit year, followed by a "/" character, and the characters "MM" are replaced with a two-digit month.

The following list describes the valid pattern letters and their meaning:

- d – Day of the month. It is interpreted as numeric in one or two digits. For example: 2 or 18
- dd – Day of the month. It is interpreted as numeric in two digits. For example: 02 or 18
- ddd – Day of the month. It is interpreted as a three letters abbreviation. For example: Wed
- dddd – Day of the month. It is interpreted as the full name. For example: Wednesday
- M – Month. It is interpreted as numeric in one or two digits. For example: 3 or 11
- MM – Month. It is interpreted as numeric in two digits. For example: 03
- MMM – Month. It is interpreted as a three letters abbreviation. For example: Mar
- MMMM – Month. It is interpreted as the full name. For example: March
- yy – Year. It is interpreted as numeric in two digits. For example: 16
- yyyy – Year. It is interpreted as numeric in four digits. For example: 2016
- h – Hour of the day in a 12-hour format [1 – 12]. It is interpreted as numeric in one or two digits. For example: 1 or 12
- hh – Hour of the day in a 12-hour format [1 – 12]. It is interpreted as numeric in two digits. For example: 01 or 12
- H – Hour of the day in a 24-hour format [0 – 23]. It is interpreted as numeric in one or two digits. For example: 0 or 23
- HH – Hour of the day in a 24-hour format [0 – 23]. It is interpreted as numeric in two digits. For example: 00 or 23
- m – Minute of the hour [0 – 59]. It is interpreted as numeric in one or two digits. For example: 0 or 59
- mm – Minute of the hour [0 – 59]. It is interpreted as numeric in two digits. For example: 00 or 59
- s – Seconds in the minute [0 – 59]. It is interpreted as numeric in one or two digits. For example: 0 or 59
- ss – Seconds in the minute [0 – 59]. It is interpreted as numeric in two digits. For example: 00 or 59
- l – Milliseconds. It is interpreted as numeric in three digits. For example: 100
- L – Milliseconds. It is interpreted as numeric in two digits (rounded, if needed). For example: 10
- t – am/pm one letter indicator. For example: a or p

- tt – am/pm two letters indicator. For example: am or pm
- T – AM/PM one letter indicator. For example: A or P
- TT – AM/PM two letters indicator. For example: AM or PM
- UTC: – indicates that UTC time zone should be used. For example: UTC:dd/MM/yyyy HH:mm:ss

Here are several common date and time formats:

- ISO date – "yyyy-MM-dd"
- ISO time – "HH:mm:ss"
- ISO date and time – "yyyy-MM-dd'T'HH:mm:ss"
- long date – "MMMM d, yyyy"
- short time – "h:mm TT"
- long time – "h:mm:ss TT"

“date string” and “datetime” data types

In addition, we would like to explain the difference between “date string” and “datetime” data types.

“date string” is a date field that can be used for rows, columns or pages in the pivot table when you want to represent date as a string. The members of such a field are formatted using datePattern and sorted inside the component as dates.

“datetime” is a date field that can be used for values in the pivot table. Min, Max, Count and Distinct Count aggregations can be applied to it. The aggregated date/time strings in values cells in the pivot table are formatted using dateTimePattern.

8.9. Calculated values

Calculated values provide you with the possibility to add some measures missing in the original data. They can be saved and restored within the report. This feature is available for JSON, CSV and OCSV data sources. Each calculated measure is described inside measure object. It can have the following parameters:

- uniqueName – measure unique name, this property will be used as an identifier for the measure inside Pivot component and as the identifier to remove calculated measure via API.
- caption (optional) – measure caption.
- formula – string that represents the formula that can contain the following operations: +, -, *, /; the following operators:isNaN(), !isNaN(); other measures can be addressed using measure unique name and aggregation function, for example sum("Price") or max("Order"). Pivot supports the following aggregation functions for CSV, OCSV and JSON data sources: "sum", "count", "distinctcount", "average", "product", "min", "max", "percent", "percentofcolumn", "percentofrow", "index", "difference", "%difference".
- grandTotalCaption (optional) – measure grand total caption.
- active (optional) – boolean value that defines whether the calculated measure will be added to the list of available values but not selected (false) or will be selected for the report (true).
- individual (optional) – Boolean. Defines whether the formula is calculated using raw values (true) or using aggregated values (false). Default value is false.
- format (optional) – number formatting name.

This example on JSFiddle(https://jsfiddle.net/flexmonster/x3qw9s71/)
illustrates how to define calculated measure with the minimal price for each color. You should define slice the following way:

```
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 calculated measure with 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: "if(#value = 1, 'trueStyle')",
        measure: "Top Category",
        trueStyle: {
            backgroundColor: "#66FF99",
            color: "#000000",
            fontFamily: "Arial",
            fontSize: 12
        }
    }
]
```

This JSFiddle(https://jsfiddle.net/flexmonster/qkhe4wkq/)
shows how you can specify number format for your calculated measure.

individual property allows calculating the formula using raw values. In the example(https://jsfiddle.net/flexmonster/7mtwxqow/)
formula sum('Price') * sum('Amount') will be calculated as following:

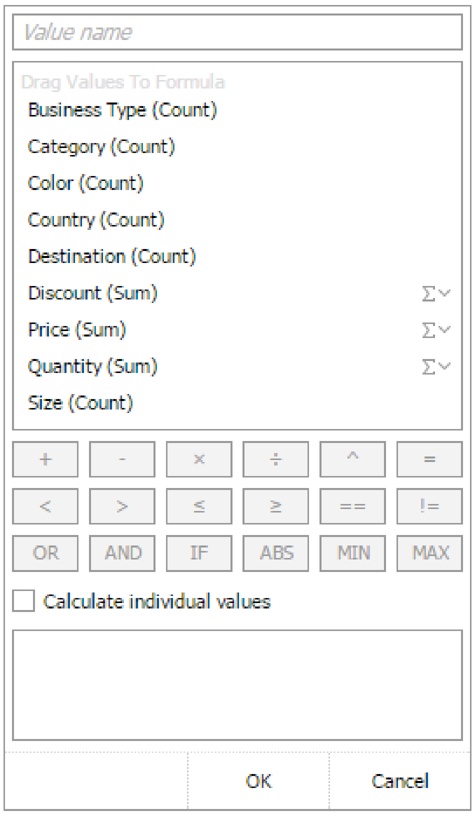
set individual: true: 174 * 36 + 225 * 44
set individual: false: (174 + 225) * (36 + 44)

The following report illustrates how to use 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
      }
  ]
 },
 options: {
  configuratorActive: false
 }
}
```

Add calculated values using Fields List

Please use Add calculated value in Fields List to add calculated measure in run time.



?alculated values via API

Calculated measures can be defined within the report or added via
addCalculatedMeasure(measure:Object)(/api/addcalculatedmeasure/)
API call. To remove calculated measure use
removeCalculatedMeasure(measureName:String)(/api/removecalculatedmeasure/)
. removeAllCalculatedMeasures()(/api/removeallcalculatedmeasures/)
removes all calculated measures.

8.10. Custom sorting

Custom sorting allows you to define a specific order for displaying columns, rows or pages. It can be set using sortOrder property:

sortOrder (optional) – Array. Using this property you can set custom sorting for hierarchy members. You can specify sortOrder the following way: [“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:

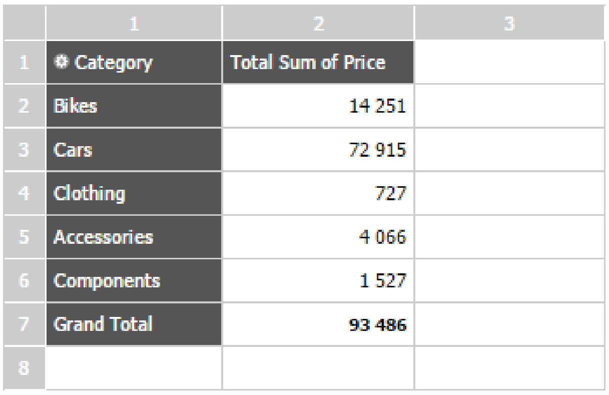
```
{
    "dataSource": {
        "filename": "http://www.flexmonster.com/download/data.csv"
    },
    "slice": {
        "rows": [
            {
                "uniqueName": "Category"
            }
        ],
        "measures": [ {"uniqueName": "Price"} ]
    }
}
```

This hierarchy has the following members: “Accessories”, “Bikes”, “Clothing”, “Components”, “Cars”. To define custom sorting we add sortOrder property:

```
{
    "dataSource": {
        "filename": "http://www.flexmonster.com/download/data.csv"
    },
    "slice": {
        "rows": [
            {
                "uniqueName": "Category",
                "sortOrder": ["Bikes", "Cars", "Clothing",
                    "Accessories", "Components"]
            }
        ],
        "measures": [ {"uniqueName": "Price"} ]
    }
}
```

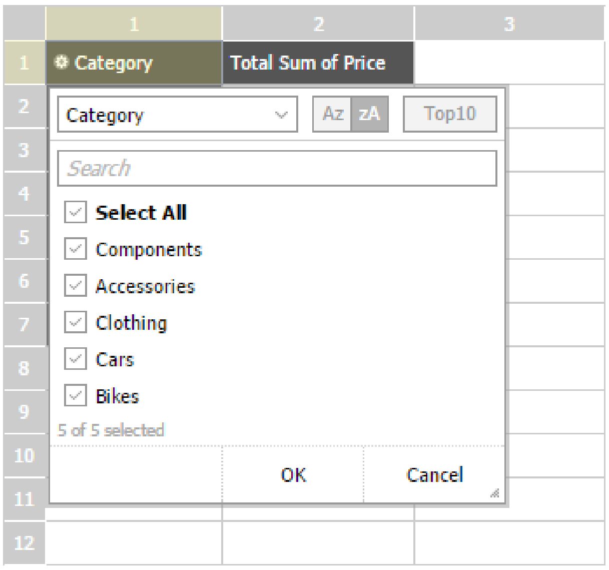
Open the example on JSFiddle(http://jsfiddle.net/flexmonster/39z1xy8y/)
.

Now hierarchy members will be displayed in predefined order:

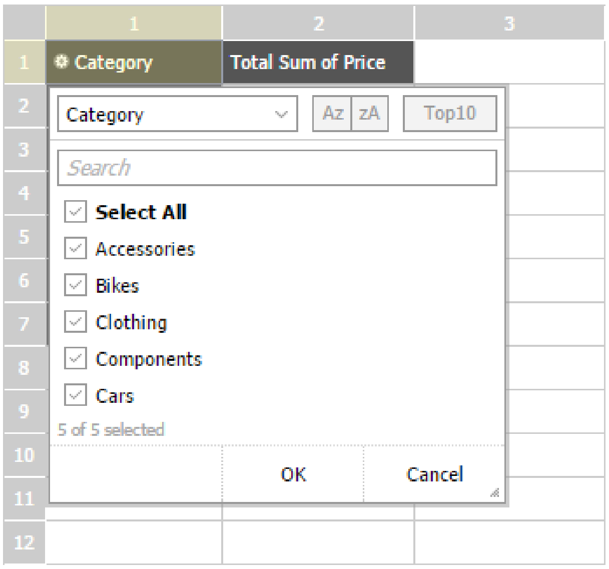


	1	2	3
1	Category	Total Sum of Price	
2	Bikes	14 251	
3	Cars	72 915	
4	Clothing	727	
5	Accessories	4 066	
6	Components	1 527	
7	Grand Total	93 486	
8			

The reverse alphabetical order will sort opposite to the order defined in sortOrder property:



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 pages in the slice.

9. Integration

Starting from version 2.3 you can easily integrate the component with popular frameworks. It will take only a few lines of code to start using Flexmonster in your JS framework.

Web Pivot Table Component can be natively used with JavaScript or TypeScript and integrates with client-side frameworks such as jQuery, Angular, React, Require, PhoneGap. Also, we have tutorials for server side technologies: ASP.NET and Java JSP.

Guides

Follow detailed tutorials for each technology:

- Integration with Angular(/doc/integration-with-angularjs/)
- Integration with Angular 2(/doc/integration-with-angular-2/)
- Integration with Angular 4(/doc/integration-with-angular-4/)
- Integration with React(/doc/integration-with-react/)
- Integration with Require(/doc/integration-with-requirejs/)
- Integration with TypeScript(/doc/integration-with-typescript/)
- Integration with ASP.NET(/doc/integration-with-asp-net/)
- Integration with JSP(/doc/integration-with-jsp/)
- Integration with PhoneGap(/doc/integration-with-phonegap/)

9.1. Integration with AngularJS

This tutorial will help you to integrate Pivot Table Component with AngularJS framework(https://angularjs.org/) . Follow the steps to set up a simple project:

Step 1: Create a new AngularJS project

1. Create a new folder for the project, i.e. my-angular-project/.
2. Copy flexmonster/ folder into your project folder. You can find flexmonster/ inside of the package you have downloaded from the site.
3. Create index.html file inside my-angular-project/ with simple angular 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 jQuery and Flexmonster libraries to the 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://code.jquery.com/jquery-2.2.4.min.js"></script>
    <script src="flexmonster/flexmonster.js"></script>
</head>
<body>
</body>
</html>
```

Step 3: Initialize Pivot Table

Add flexmonster module to App and use fm-pivot directive to add FM pivot table to the index.html. After that, an empty grid will be shown. Please note, fm-license-key is your license or trial key, so you should replace XXX with an actual key:

```
<!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://code.jquery.com/jquery-2.2.4.min.js"></script>
    <script src="flexmonster/flexmonster.js"></script>
</head>
<body>

<div fm-pivot
     fm-toolbar="true"
     fm-license-key="XXX">
</div>

</body>
</html>
```

Step 4: Load sample report

To see some data on grid add fm-report attribute with report 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://code.jquery.com/jquery-2.2.4.min.js"></script>
    <script src="flexmonster/flexmonster.js"></script>
</head>
<body>
<div fm-pivot
     fm-toolbar="true"
     fm-report="'http://www.flexmonster.com/download/report.xml'"
     fm-license-key="XXX">
</div>
</body>
</html>
```

Properties

Please note, that every attribute for fm-pivot directive is set either as a string value or as an angular variable. List of available attributes:

- fm-toolbar – parameter to embed the toolbar or not. Default value is false – without the toolbar.
- fm-license-key – the license key.
- fm-width – width of the component on the page (pixels or percent). The default value for width is 100%.
- fm-height – height of the component on the page (pixels or percent). The default value for height is 500.
- fm-component-folder – URL of the component's folder which contains all necessary files. Also, it is used as a base URL for report files, localization files, styles and images. The default value for componentFolder is flexmonster/.
- fm-report – property to set a report. It can be inline report JSON, URL to report JSON or URL to report XML.
- fm-global – object that allows you to preset options for all reports. These options can be overwritten for concrete reports. Object structure is the same as for report.

Event handlers for fm-pivot directive:

- fm-ready
- fm-report-complete
- fm-report-change
- fm-update
- fm-cell-click
- fm-cell-doubleclick
- fm-filter-open
- fm-fields-list-open
- fm-fields-list-close

All attributes are equal to those which are passed to $.flexmonster(). The only difference is that fm- prefix was added to each of them.

Examples

Please find more examples on GitHub(https://github.com/flexmonster/pivot-angularjs)
of Pivot Table Component integration with AngularJS framework.

9.2. Integration with Angular 2

This tutorial will help you to integrate Pivot Table Component with Angular 2 framework(https://angular.io/)
. It is based on angular.io quickstart(https://angular.io/docs/ts/latest/quickstart.html)
.

Prerequisites

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

Open 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.

Create a new project based on this sample

Download .zip archive with the sample or clone it from GitHub(https://github.com/flexmonster/pivot-angularjs2)
 within the following command:

```
git clone https://github.com/flexmonster/pivot-angularjs2 my-proj
cd my-proj
```

Install npm packages

Install the npm packages described in the package.json and verify that it works:

```
npm install
npm start
```

The npm start command first compiles the application, then simultaneously recompiles and runs the lite-server. Both the compiler and the server watch for file changes.

Shut it down manually with Ctrl-C.

You’re ready to write your application.

Project structure

Let’s overview the sample folder structure:

- app/ – folder that contains application files:
  - app.module.ts – the file imports FlexmonsterPivot module for further usage in Angular 2 application.
  - app.component.html – main component view, here we insert the instance of the pivot table. It uses <fm-pivot> directive which was made to integrate Flexmonster with Angular 2.
  - app.component.ts – main component code, recommended place to use Flexmonster API. Inside our sample project we show how to handle ready event.
  - flexmonster.angular2.d.ts – declares FlexmonsterPivot module.
  - flexmonster.d.ts – TypeScript definitions for Flexmonster.
  - flexmonster.angular2.ts – the code of <fm-pivot> directive.
- index.html – the starting point for Angular 2 project. In this file we add Flexmonster and jQuery

dependencies the following way:

```
<script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
<script src="https://cdn.flexmonster.com/2.3/flexmonster.js"></script>
```

In our sample, we are loading flexmonster.js and jquery.js from CDN. You can also download(/download-page/) all Flexmonster files and copy flexmonster/ folder to your project.

Please find contents of key files below.

app.component.html

```
<h1>AngularJS 2/Flexmonster</h1>
<fm-pivot [componentFolder]="'https://cdn.flexmonster.com/2.3/'"
          [toolbar]="true"
          [width]="'100%'"
          [height]="500"
          [licenseKey]="'XXXX-XXXX-XXXX-XXXX-XXXX'"
          [report]="'https://cdn.flexmonster.com/2.3/reports/report.json'"
          (ready)="onPivotReady($event)">
    Flexmonster will appear here
</fm-pivot>
```

app.component.ts

```
import { Component } from '@angular/core';
import { FlexmonsterPivot } from "./flexmonster.angular2";

@Component({
    selector: 'my-app',
    templateUrl: './app/app.component.html'
})
export class AppComponent {
    onPivotReady(pivot: Flexmonster.Pivot): void {
        console.log("[ready] FlexmonsterPivot");
    }
}
```

app.module.ts

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

import { AppComponent }  from './app.component';
import { FlexmonsterPivot } from "./flexmonster.angular2";

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

9.3. Integration with Angular 4

This tutorial will help you to integrate Pivot Table Component with Angular 4 framework(https://angular.io/)
. It is based on angular.io quickstart(https://angular.io/docs/ts/latest/quickstart.html)
.

Prerequisites

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

Open 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.

Create a new project based on this sample

Download .zip archive with the sample or clone it from GitHub(https://github.com/flexmonster/pivot-angularjs4)
 within the following command:

```
git clone https://github.com/flexmonster/pivot-angularjs4 my-proj
cd my-proj
```

Install npm packages

Install the npm packages described in the package.json and verify that it works:

```
npm install
npm start
```

The npm start command first compiles the application, then simultaneously recompiles and runs the lite-server. Both the compiler and the server watch for file changes.

Shut it down manually with Ctrl-C.

You’re ready to write your application.

Project structure

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

- app/ – folder containing application files:
  - app.module.ts – here we import FlexmonsterPivot module in order to use it in the Angular 4 application.
  - app.component.html – in this file we embed our pivot table. A special directive <fm-pivot> is used to integrate Flexmonster pivot table with Angular 4.
  - app.component.ts – we recommend to use Flexmonster API in this file. As an example, we are handling ready event.
  - flexmonster.angular4.d.ts – declares FlexmonsterPivot module.
  - flexmonster.d.ts – TypeScript definitions for Flexmonster.
  - flexmonster.angular4.ts – the code of <fm-pivot> directive.
- index.html – the entering point for Angular 4 project. We used this file to add Flexmonster and jQuery dependencies:

  ```
  <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
  <script src="https://cdn.flexmonster.com/2.3/flexmonster.js"></script>
  ```

The dependencies are using flexmonster.js and jquery.js from CDN. Another option is to download Flexmonster component(/download-page/)
and copy flexmonster/ folder to the project.

Please have a look at the content of the key files:

app.component.html

```
<h1>AngularJS 4/Flexmonster</h1>
<fm-pivot [componentFolder]="'https://cdn.flexmonster.com/2.3/'"
          [toolbar]="true"
          [width]="'100%'"
          [height]="500"
          [licenseKey]="'XXXX-XXXX-XXXX-XXXX-XXXX'"
          [report]="'https://cdn.flexmonster.com/2.3/reports/report.json'"
          (ready)="onPivotReady($event)">
    Flexmonster will appear here
</fm-pivot>
```

app.component.ts

```
import { Component } from '@angular/core';
import { FlexmonsterPivot } from "./flexmonster.angular4";

@Component({
    selector: 'my-app',
    templateUrl: './app.component.html'
})
export class AppComponent {
    onPivotReady(pivot: Flexmonster.Pivot): void {
```

```
        console.log("[ready] FlexmonsterPivot");
    }
}
```

app.module.ts

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

import { AppComponent }  from './app.component';
import { FlexmonsterPivot } from "./flexmonster.angular4";

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

9.4. Integration with React

This tutorial will help you to integrate Pivot Table Component with React framework(https://facebook.github.io/react/)
. Follow the steps to set up a simple React/JSX app using Flexmonster Pivot Table:

Step 1: Create a new React project

1. Create a new folder for the project, i.e. my-react-project/.
2. Copy flexmonster/ folder into your project folder. You can find flexmonster/ inside of the package you have downloaded from the site.
3. Create index.html file inside my-react-project/ and add React dependencies:

```
<!DOCTYPE html>
<html>
<head>
    <title>My React/Flexmonster Project</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.1.0/react.js"></scr
ipt>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.1.0/react-
dom.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.24/browser.min.js"></script>
</head>
<body>
</body>
```

```
</html>
```

Step 2: Add Flexmonster dependencies

Add jQuery and Flexmonster libraries to the index.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>My React/Flexmonster Project</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.1.0/react.js"></scr
ipt>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.1.0/react-
dom.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.24/browser.min.js"></script>
    <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
    <script src="flexmonster/flexmonster.js"></script>
</head>
<body>
</body>
</html>
```

Step 3: Initialize Pivot Table

1. Add div to insert Pivot:

```
<body>
    <div id="fm-container"></div>
</body>
```

1. Render pivot to fm-container using React. After that an empty grid will be shown. Please note, licenseKey is your license or trial key, so you should replace XXX with an actual key:

```
<script type="text/babel">
 ReactDOM.render(
  <FlexmonsterReact.Pivot licenseKey="XXX"/>,
  document.getElementById("fm-container")
 );
</script>
```

Step 4: Load sample report

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

```
<script type="text/babel">
 ReactDOM.render(
  <FlexmonsterReact.Pivot
report="http://www.flexmonster.com/download/report.xml" licenseKey="XXX"/>,
  document.getElementById("fm-container")
 );
</script>
```

Properties

All available attributes for FlexmonsterReact.Pivot the same as for $.flexmonster().

- toolbar – parameter to embed the toolbar or not. Default value is false – without the toolbar.
- licenseKey – the license key.
- width – width of the component on the page (pixels or percent). The default value for width is 100%.
- height – height of the component on the page (pixels or percent). The default value for height is 500.
- componentFolder – URL of the component's folder which contains all necessary files. Also, it is used as a base URL for report files, localization files, styles and images. The default value for componentFolder is flexmonster/.
- report – property to set a report. It can be inline report JSON, URL to report JSON or URL to report XML.
- global – object that allows you to preset options for all reports. These options can be overwritten for concrete reports. Object structure is the same as for report.

Event handlers:

- ready
- reportcomplete
- reportchange
- update
- cellclick
- celldoubleclick
- filteropen
- fieldslistopen
- fieldslistclose

Examples

Please find more examples on GitHub(https://github.com/flexmonster/pivot-react)
 of Pivot Table Component integration with React framework.

9.5. Integration with RequireJS

This tutorial will help you to integrate Pivot Table Component with RequireJS framework(http://requirejs.org/)
. The main purpose of this tutorial is to show the process of setting up RequireJS project with jQuery and Flexmonster. Our sample is based on the jquery shim sample that is provided in the RequireJS docs(http://requirejs.org/docs/jquery.html)
.

Project structure

It's important to follow the project structure and configure shim for jQuery and Flexmonster. Let's overview the sample folder structure:

- js/ – folder that contains all JavaScript files

    - app/ – folder that contains application JS files
        - main.js – main application JavaScript file
    - lib/ – folder that contains project dependencies
        - flexmonster/ – folder with Flexmonster pivot table files
        - jquery.js – jQuery library
        - require.js – RequireJS library
    - app.js – main RequireJS file, contains configuration
- app.html – main HTML file, the entry point for your project

Please find contents of key files below.

app.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>jQuery+RequireJS Sample Page</title>
        <script data-main="js/app" src="js/lib/require.js"></script>
    </head>
    <body>
        <h1>jQuery+RequireJS+Flexmonster Sample Page</h1>
        <div id="fm-container"></div>
    </body>
</html>
```

js/app.js

```
/* Place third party dependencies in the lib folder.
Configure loading modules from the lib directory, except 'app' ones. */
requirejs.config({
    "baseUrl": "js/lib",
    "paths": {
      "app": "../app"
    },
    "shim": {
        "flexmonster/flexmonster": ["jquery"]
    }
});

/* Load the main app module to start the app */
requirejs(["app/main"]);
```

js/app/main.js

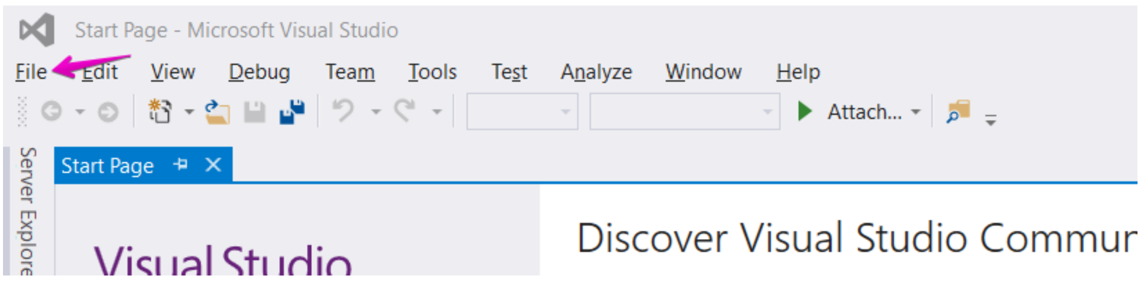
```
/* Please note, licenseKey is your license or trial key,
so you should replace XXX with an actual key. */
define(["jquery", "flexmonster/flexmonster"], function($) {
    $("#fm-container").flexmonster({
     componentFolder: "js/lib/flexmonster/",
     width: "100%",
     height: "500",
     toolbar: true,
     licenseKey: "XXX",
     report: "http://www.flexmonster.com/download/report.xml"
    });
});
```

9.6. Integration with TypeScript

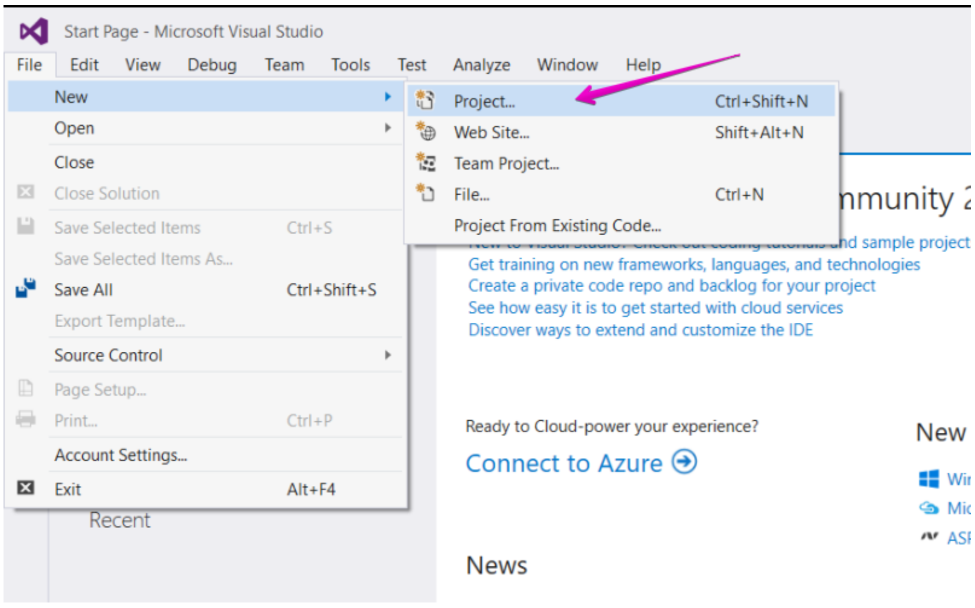
This tutorial will help you to integrate Pivot Table Component with TypeScript(https://www.typescriptlang.org/)
. Follow the steps to set up a simple TypeScript project in Visual Studio using Flexmonster Pivot Table:

Step 1. Create new TypeScript project in Visual Studio

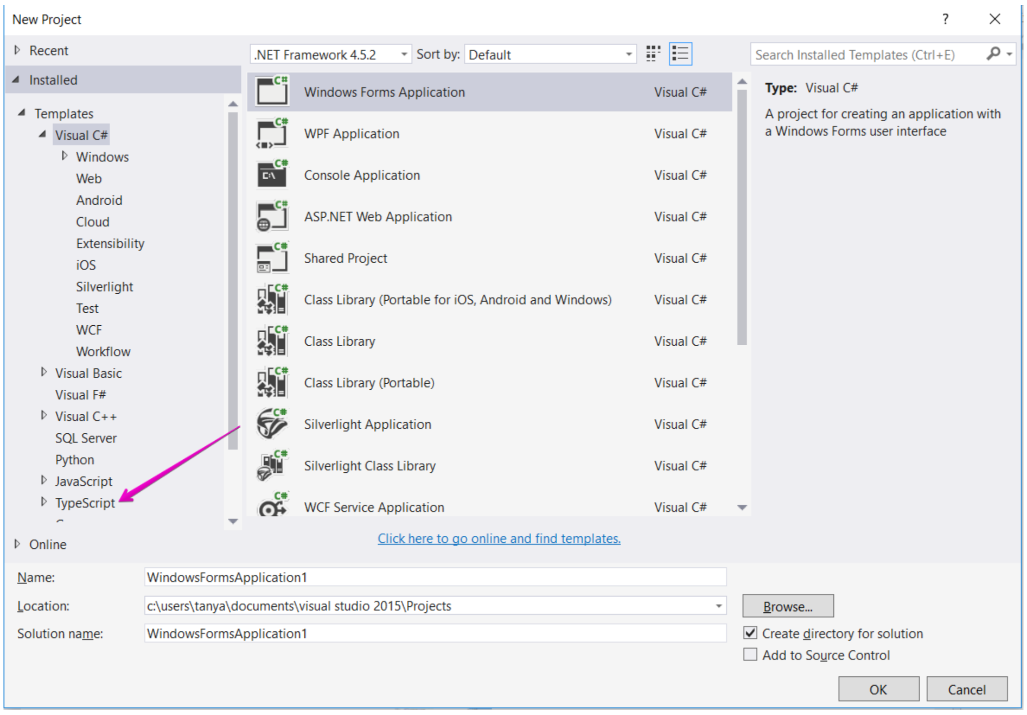
1. Open Visual Studio. If you don’t have any, we recommend Visual Studio 2015(https://www.visualstudio.com/vs/)
   .



2. Click File.


Start Page - Microsoft Visual Studio
File Edit View Debug Team Tools Test Analyze Window Help
New
Open
Close
Close Solution
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Export Template...
Source Control
Page Setup...
Print... Ctrl+P
Account Settings...
Exit Alt+F4
Project... Ctrl+Shift+N
Web Site... Shift+Alt+N
Team Project...
File... Ctrl+N
Project From Existing Code...
Get training on new frameworks, languages, and technologies
Create a private code repo and backlog for your project
See how easy it is to get started with cloud services
Discover ways to extend and customize the IDE
Ready to Cloud-power your experience?
Connect to Azure
Recent
News
New

3. Choose New -> Project.



New Project ? ×

Recent
Installed
Templates
Visual C#
Windows
Web
Android
Cloud
Extensibility
iOS
Silverlight
Test
WCF
Workflow
Visual Basic
Visual F#
Visual C++
SQL Server
Python
JavaScript
TypeScript
Online

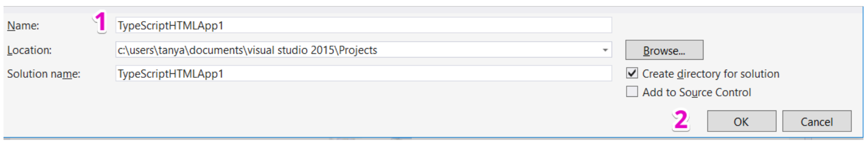
.NET Framework 4.5.2 Sort by: Default Search Installed Templates (Ctrl+E)

Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
ASP.NET Web Application	Visual C#
Shared Project	Visual C#
Class Library (Portable for iOS, Android and Windows)	Visual C#
Class Library	Visual C#
Class Library (Portable)	Visual C#
Silverlight Application	Visual C#
Silverlight Class Library	Visual C#
WCF Service Application	Visual C#

Type: Visual C#
A project for creating an application with a Windows Forms user interface

Click here to go online and find templates.

Name: WindowsFormsApplication1
Location: c:\users\tanya\documents\visual studio 2015\Projects Browse...
Solution name: WindowsFormsApplication1
☑ Create directory for solution
☐ Add to Source Control
OK Cancel

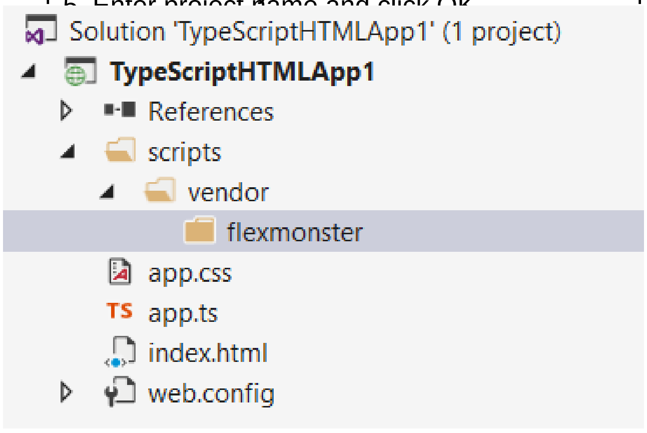


Name: 1 TypeScriptHTMLApp1
Location: c:\users\tanya\documents\visual studio 2015\Projects Browse...
Solution name: TypeScriptHTMLApp1
☑ Create directory for solution
☐ Add to Source Control
2 OK Cancel

Step 2. Add Flexmonster files

Create folder scripts/vendor/ inside your project. Copy flexmonster/ folder from the download package to scripts/vendor/. Here you can also add other JavaScript libraries for your project.

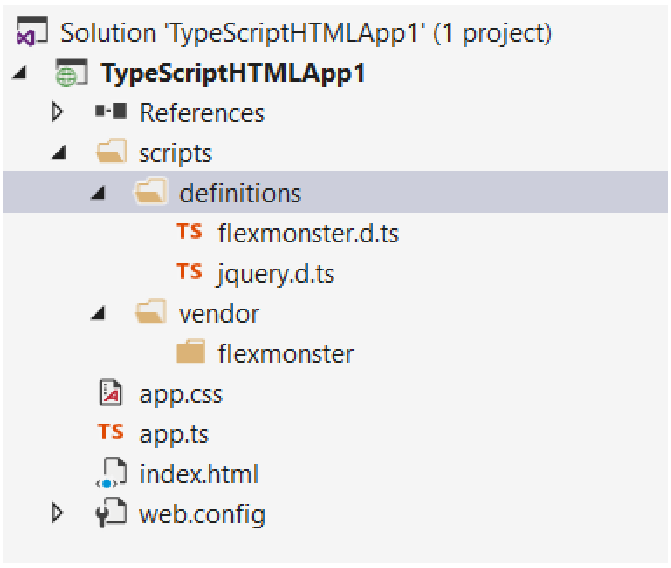
5. Enter project name and click Ok.



Solution 'TypeScriptHTMLApp1' (1 project)
TypeScriptHTMLApp1
References
scripts
vendor
flexmonster
app.css
app.ts
index.html
web.config

Step 3. Add definitions (d.ts)

Create folder scripts/definitions/ and add flexmonster.d.ts(https://github.com/flexmonster/typescript-definitions/blob/master/flexmonster.d.ts)
and jquery.d.ts(https://github.com/DefinitelyTyped/DefinitelyTyped/blob/master/jquery/jquery.d.ts)
inside. Here you can also add other definition files for your project.



Step 4. Add main TypeScript file

Create folder src/ and add new TypeScript file named “PivotApp”. Please note, licenseKey is your license or trial key, so you should replace XXX with an actual key.

```
class PivotApp {
    private pivot: Flexmonster;

    constructor() {
        this.pivot = $("#content").flexmonster({
            componentFolder: "scripts/vendor/flexmonster/",
            toolbar: true,
            licenseKey: "XXX",
            width: "100%",
            height: "500",
            ready: () => this.onPivotReady()
        });
    }

    private onPivotReady(): void {
        this.pivot.load("http://www.flexmonster.com/download/report.xml");
    }
}

window.onload = () => {
    new PivotApp();
};
```

Step 5. Add index.html file

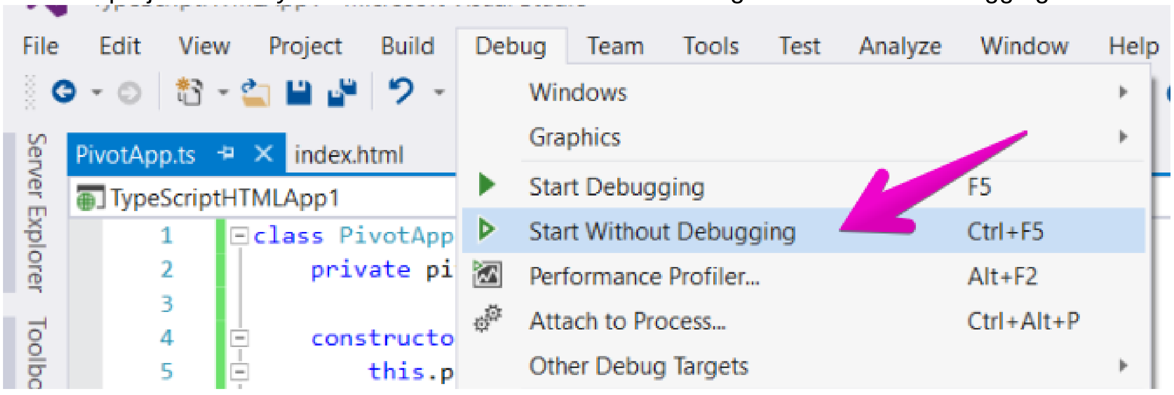
Create index.html file and add links to jquery, flexmonster and PivotApp JavaScript files.

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

```
    <meta charset="utf-8" />
    <title>TypeScript HTML App</title>
    <script src="scripts/vendor/flexmonster/lib/jquery.min.js"></script>
    <script src="scripts/vendor/flexmonster/flexmonster.js"></script>
    <script src="src/PivotApp.js"></script>
</head>
<body>
    <h1>TypeScript HTML App</h1>
    <div id="content"></div>
</body>
</html>
```

Step 6. Run the project

Now the project is ready to run. In Visual Studio choose Debug->Start without debugging.



```
class PivotApp
    private pi

    constructo
        this.p
```

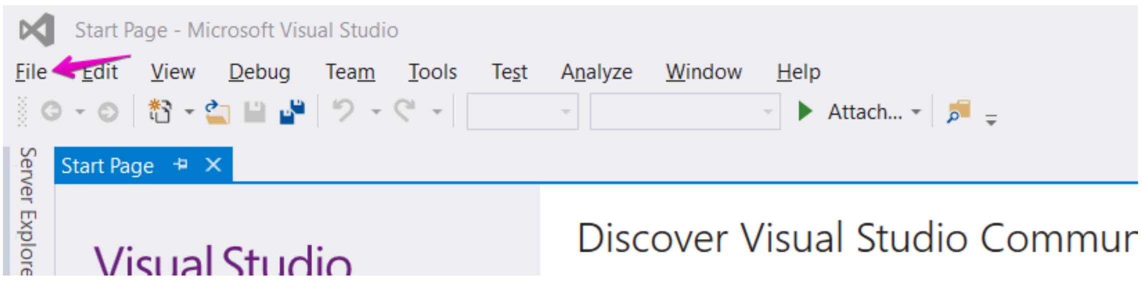
You will see Pivot Table with sample report loaded.

9.7. Integration with ASP.NET

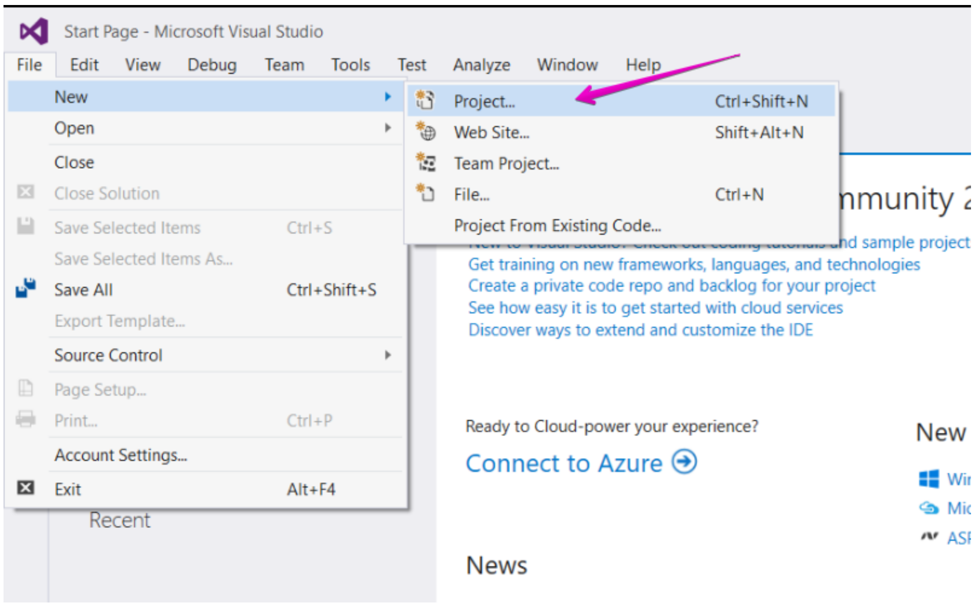
This tutorial will help you to integrate Pivot Table Component with ASP.NET(https://www.asp.net/)
. Follow the steps to set up a simple ASP.NET project in Visual Studio using Flexmonster Pivot Table:

Step 1. Create ASP.NET MVC project

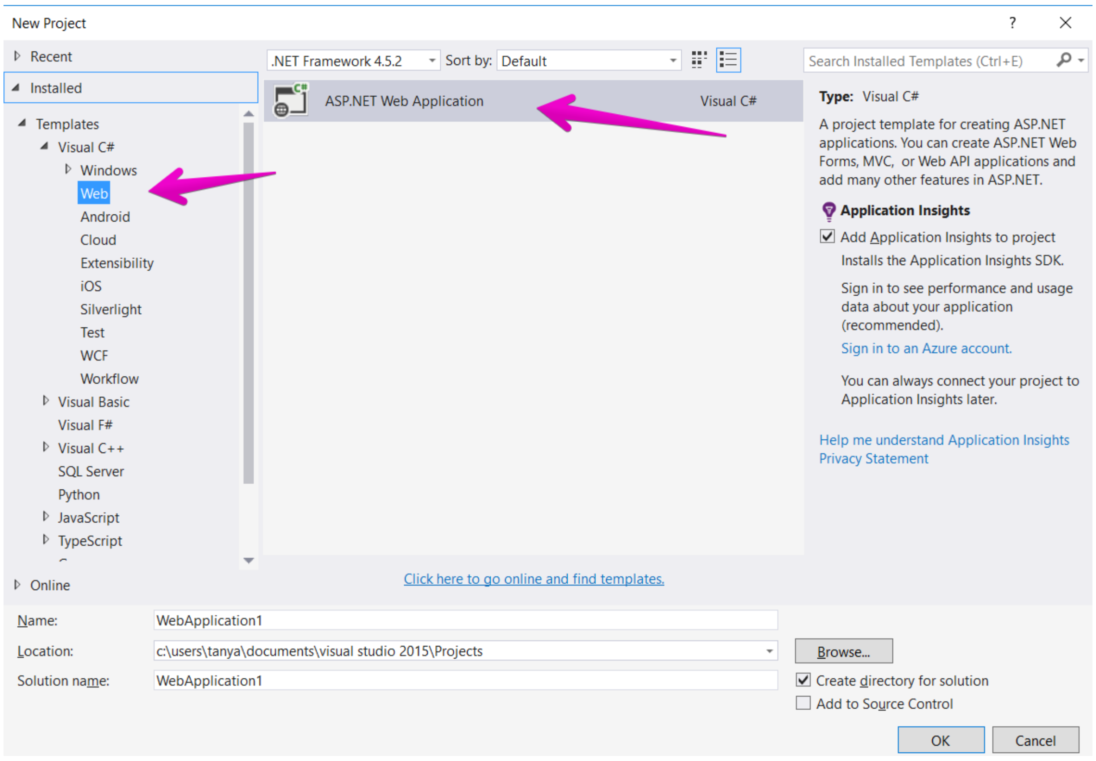
1. Open Visual Studio. If you don’t have any, we recommend Visual Studio 2015(https://www.visualstudio.com/vs/)
   .



2. Click File.



3. Choose New -> Project.



.NET Framework 4.5.2 Sort by: Default

ASP.NET Web Application Visual C#

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

☑ Add Application Insights to project

Installs the Application Insights SDK.

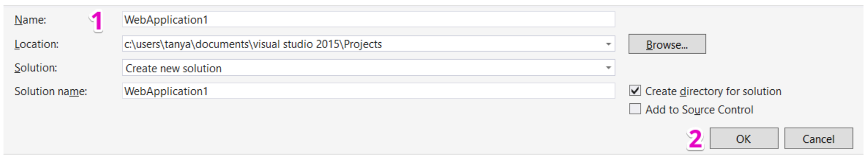
Sign in to see performance and usage data about your application (recommended).

Sign in to an Azure account.

You can always connect your project to Application Insights later.

Help me understand Application Insights
Privacy Statement

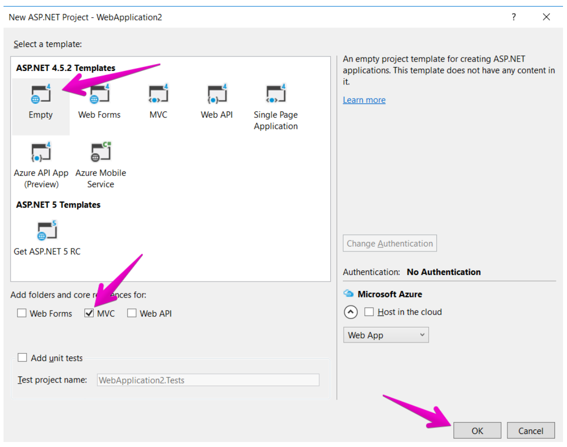
Name: WebApplication1
Location: c:\users\tanya\documents\visual studio 2015\Projects
Solution name: WebApplication1
☑ Create directory for solution
☐ Add to Source Control



Name: WebApplication1
Location: c:\users\tanya\documents\visual studio 2015\Projects
Solution: Create new solution
Solution name: WebApplication1
☑ Create directory for solution
☐ Add to Source Control

5. Enter project name and click OK.

4. Choose Web.



An empty project template for creating ASP.NET applications. This template does not have any content in it.

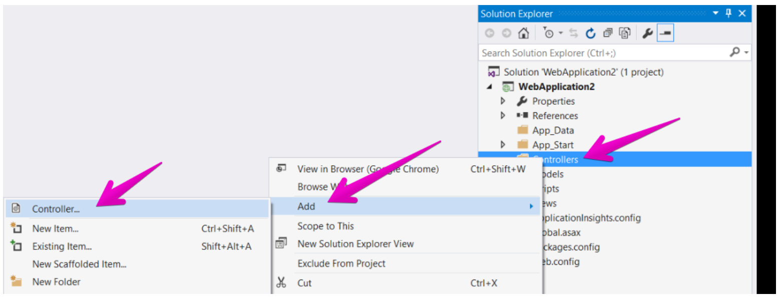
Add folders and core references for:
Web Forms [✓ MVC] Web API

Add unit tests

Test project name: WebApplication2.Tests

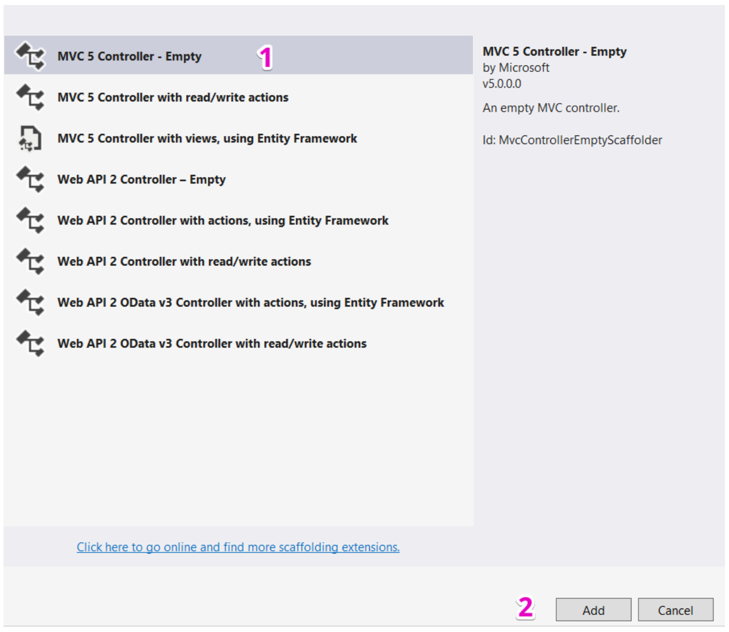
Authentication: No Authentication

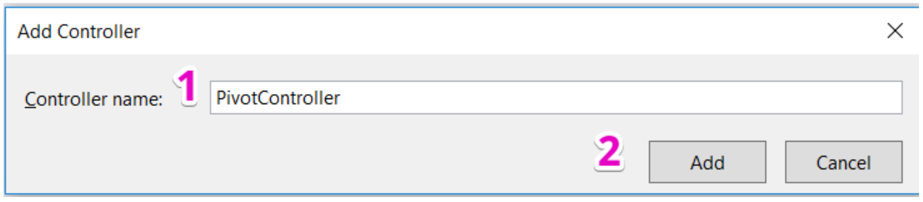
Step 2. Add View and Controller



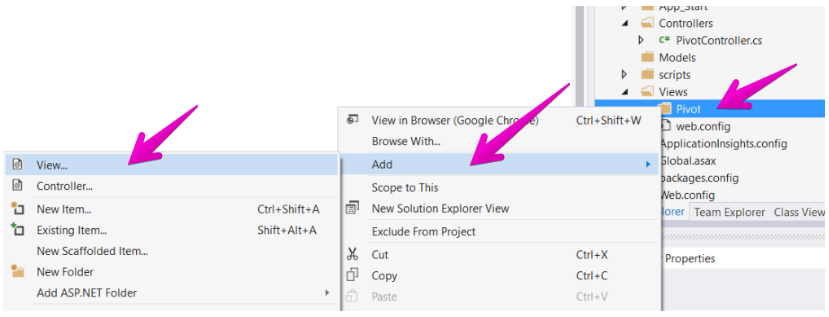
1. Choose Controllers -> Add -> Controller.

6. Select Empty template, add MVC references and click OK.


MVC 5 Controller - Empty
1
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions
MVC 5 Controller - Empty
by Microsoft
v5.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder
Click here to go online and find more scaffolding extensions.
2
Add
Cancel

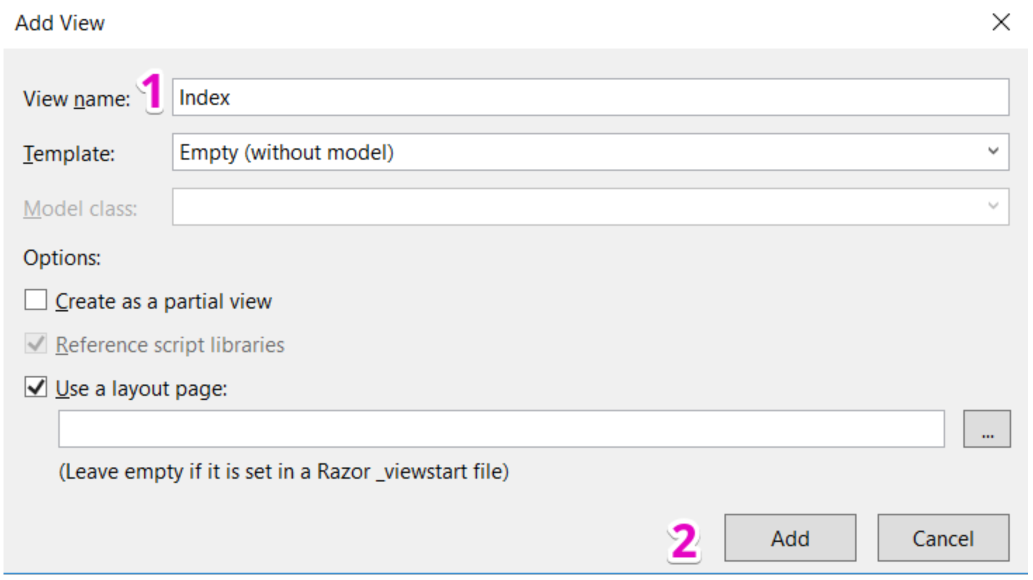


3. Enter name: PivotController.



2. Choose MVC Controller Empty and click OK.

4. Choose Views/Pivot -> Add -> View.

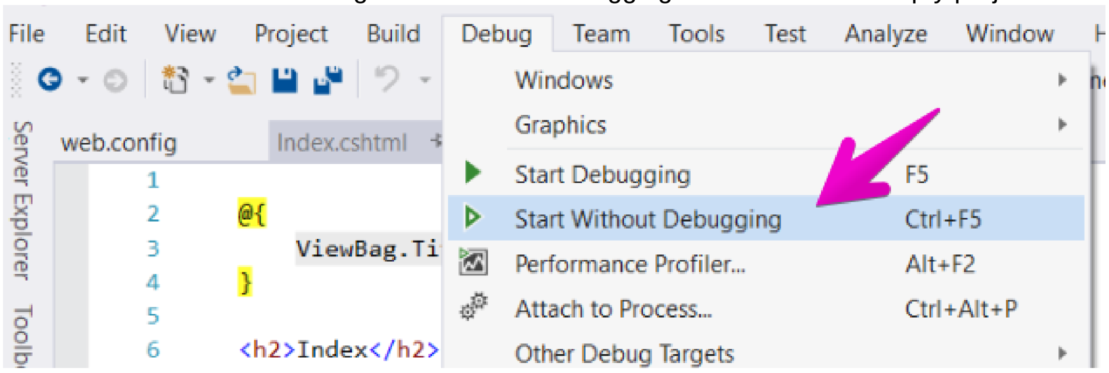


Add View

View name: Index
Template: Empty (without model)
Model class:
Options:
☐ Create as a partial view
☑ Reference script libraries
☑ Use a layout page:
(Leave empty if it is set in a Razor _viewstart file)

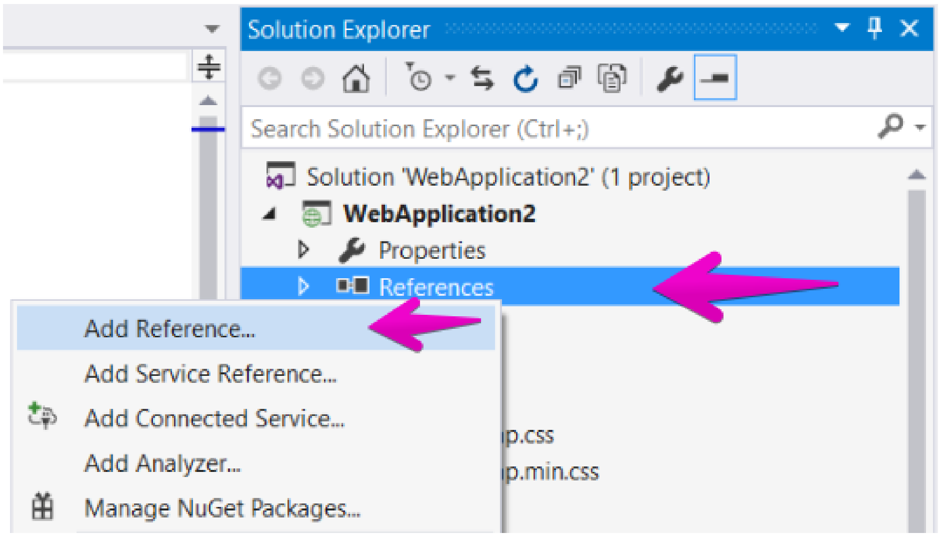
6. In Visual Studio choose Debug->Start without debugging. You will see an empty project.

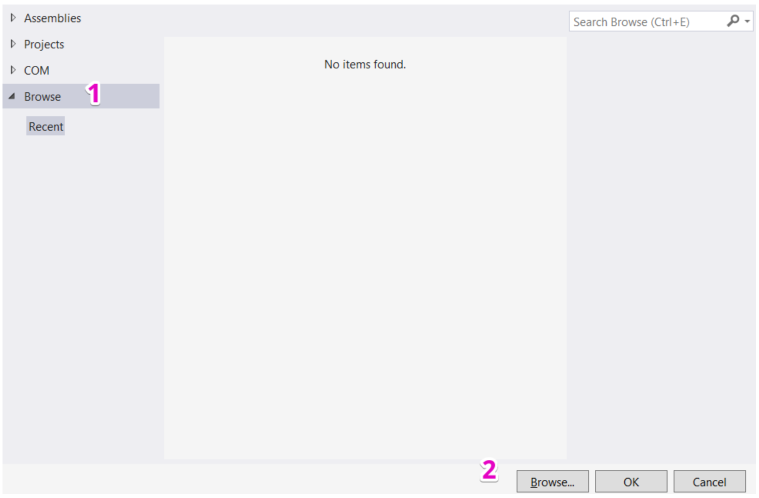
5.



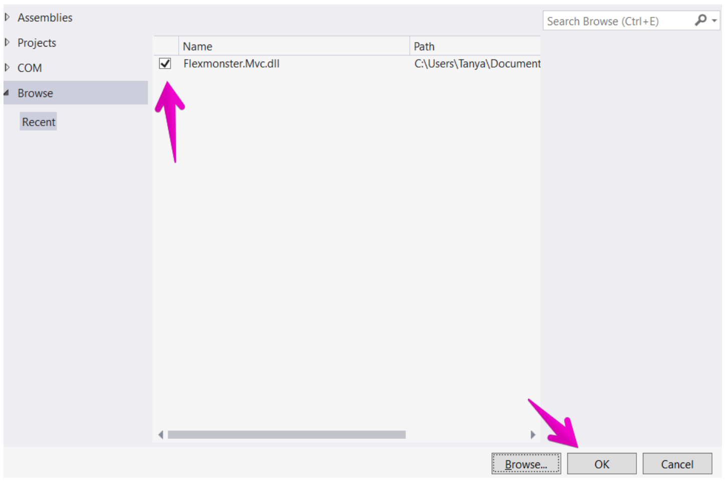
Step 3. Add Flexmonster.Mvc.dll

1. Copy Flexmonster.Mvc.dll(https://github.com/flexmonster/pivot-asp.net/blob/master/dist/Flexmonster.Mvc.dll) to your project folder.
2. Choose References -> Add reference.





4. Inside file explorer find and select Flexmonster.Mvc.dll from your project folder.



6. Open web.config file from Views/ folder. Find the following code:

3. Choose Browse and click ‘Browse’ button.

```
<namespaces>
        <add namespace="System.Web.Mvc" />
        <add namespace="System.Web.Mvc.Ajax" />
        <add namespace="System.Web.Mvc.Html" />
        <add namespace="System.Web.Routing" />
        <add namespace="WebApplication1" />
</namespaces>
```

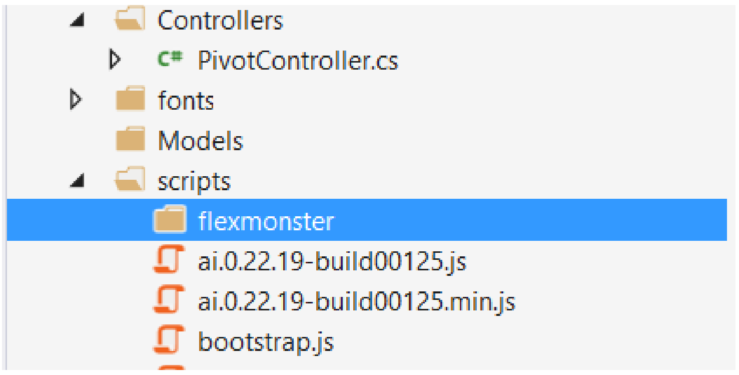
and add this line inside:

```
<add namespace="Flexmonster.Mvc" />
```

Step 4. Add Flexmonster js files to Scripts folder

5. Make sure you selected the file and click OK.

Copy flexmonster/ folder from the download package to scripts/ folder.



Step 5. Add pivot

Change content of Index.cshtml inside of Views/Pivot/. Please note, licenseKey is your license or trial key, so you should replace XXX with an actual key.

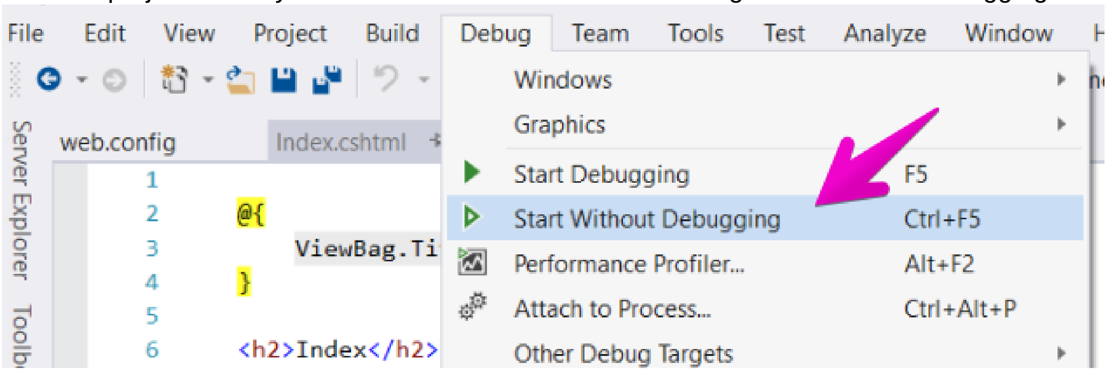
```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<script src="~/Scripts/jquery-1.10.2.min.js"></script>
<script src="~/Scripts/flexmonster/flexmonster.js"></script>
@Html.Flexmonster("pivot").Init(new Flexmonster.Mvc.Config()
{
    componentFolder = "/Scripts/flexmonster/",
    licenseKey = "XXX",
    toolbar = true,
    report = "http://www.flexmonster.com/download/report.xml"
})
```

Step 6. Build and run

Now the project is ready to run. In Visual Studio choose Debug->Start without debugging.



```
@{
    ViewBag.Ti
}
<h2>Index</h2>
```

You will see Pivot Table with sample report loaded.

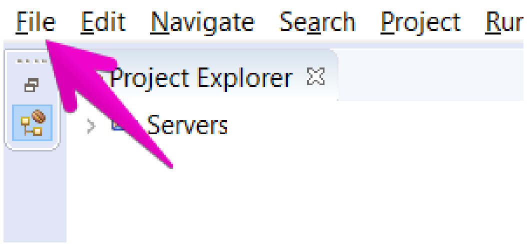
You can download the full example here(https://github.com/flexmonster/pivot-asp.net)
.

9.8. Integration with JSP

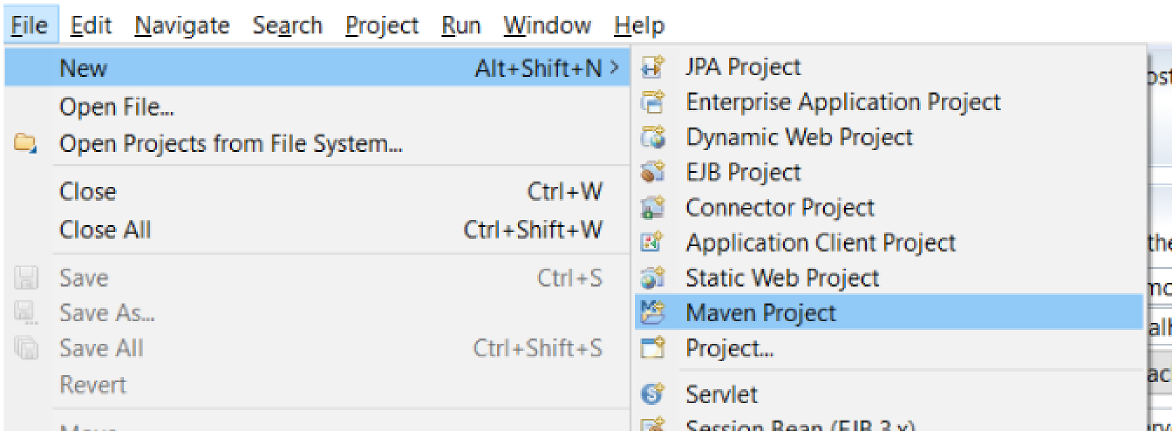
This tutorial will help you to integrate Pivot Table Component with Java JSP(http://docs.oracle.com/javaee/5/tutorial/doc/bnagy.html)
. Follow the steps to set up a simple Java JSP project with Flexmonster using Maven and Eclipse EE:

Step 1. Create new Dynamic Web Project

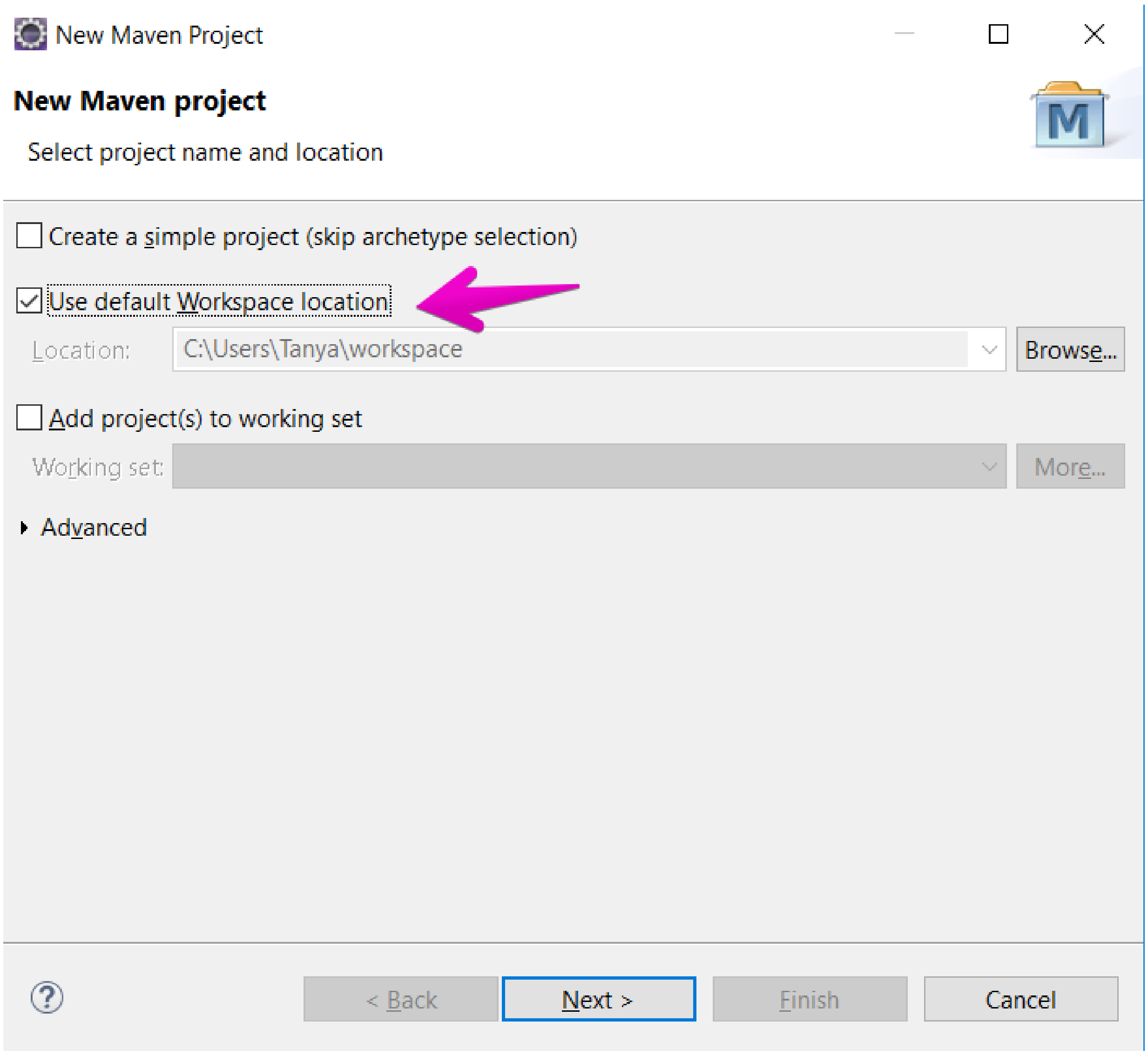
Before creating a new project you should set up Maven and Apache Tomcat Server in your Eclipse Environment.



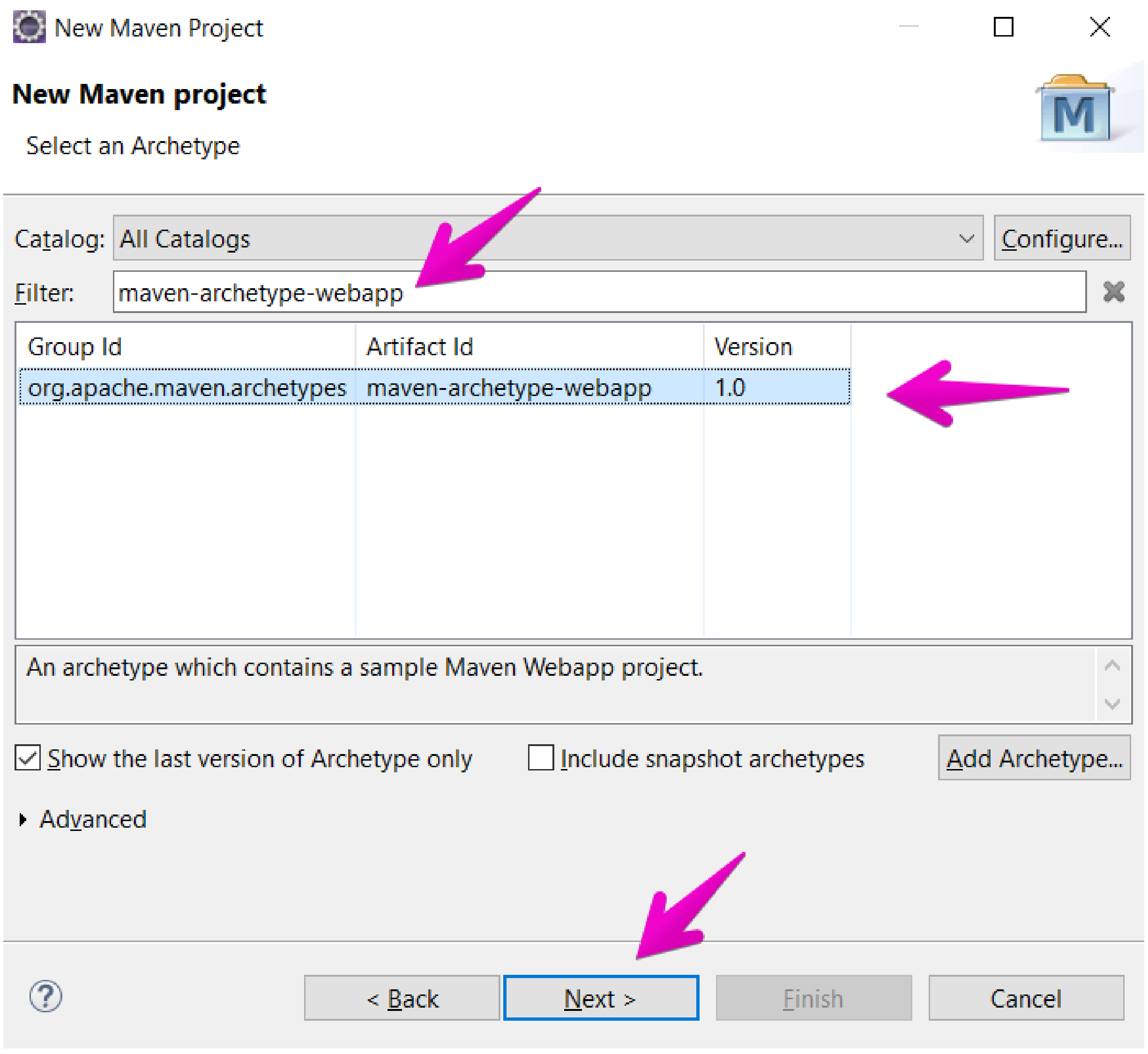
1. Click File.
2. Create a simple Maven Project in Eclipse.



3. Select default Workspace location.


New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
C:\Users\Tanya\workspace
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Finish
Cancel

4. Select the Maven archetype as: maven-archetype-webapp and click on next.



New Maven Project

New Maven project

Select an Archetype

Catalog: All Catalogs

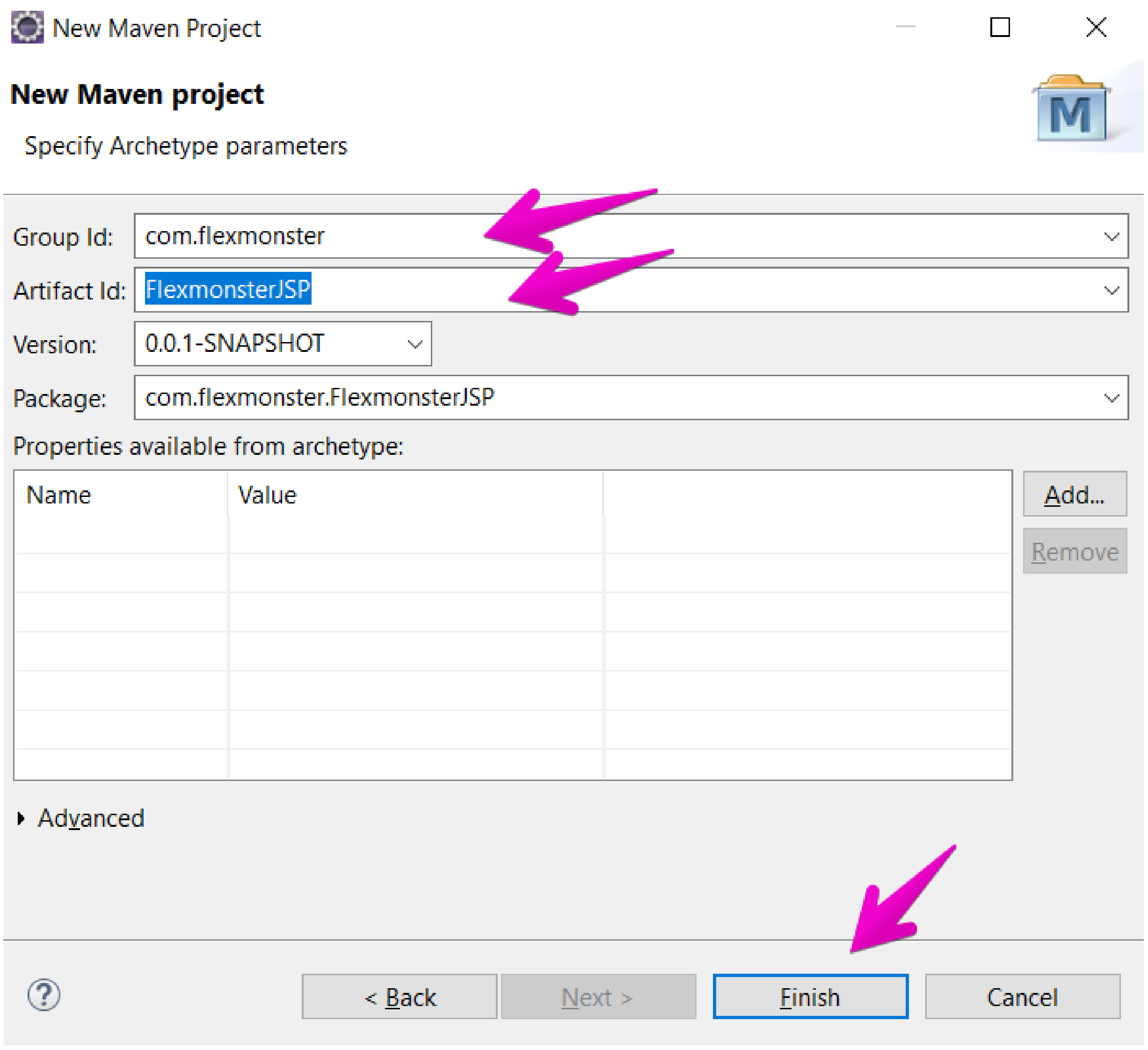
Filter: maven-archetype-webapp

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-webapp	1.0

An archetype which contains a sample Maven Webapp project.

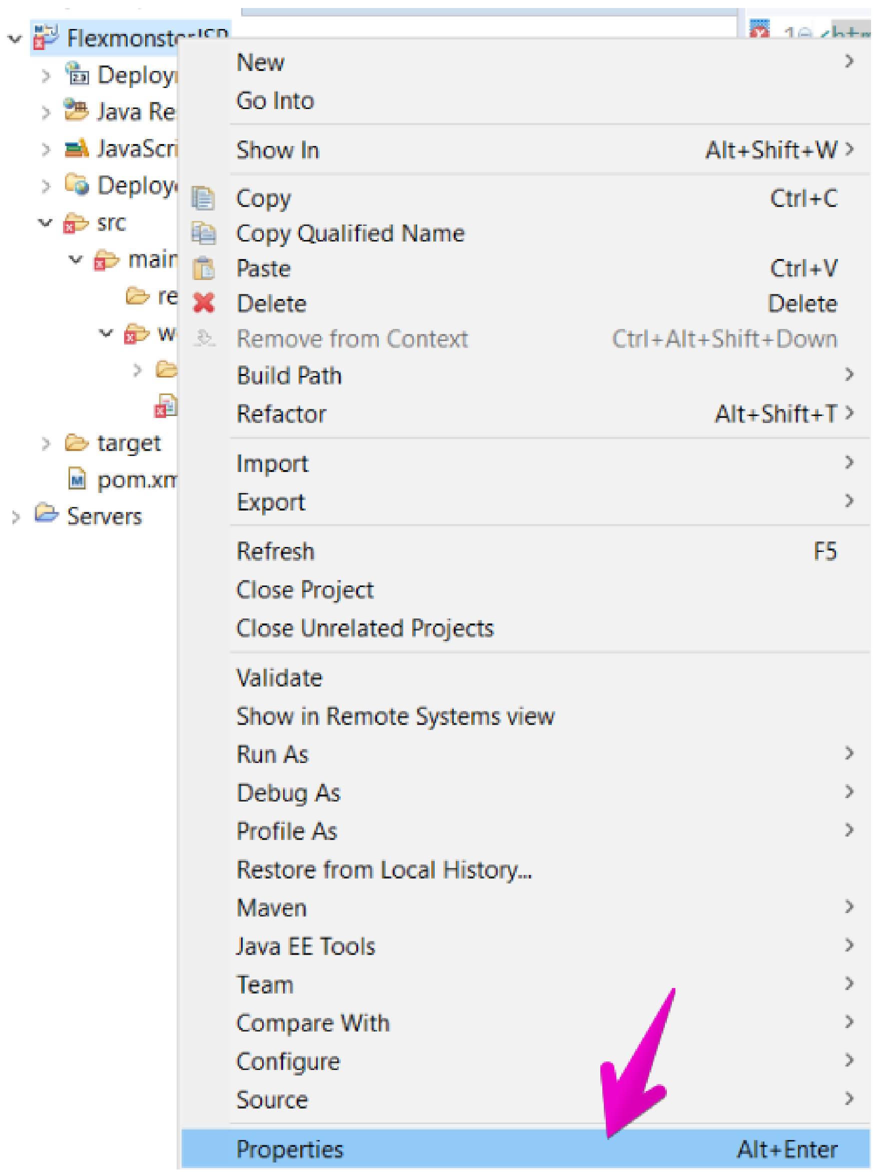
[✓] Show the last version of Archetype only
[ ] Include snapshot archetypes

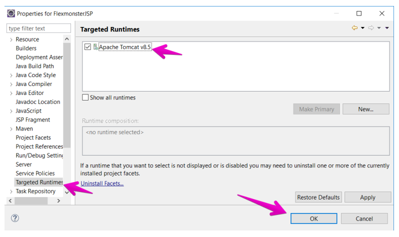
5. Fill out details below and click Finish. This step creates Maven Project in your Eclipse Environment.



Group Id: com.flexmonster
Artifact Id: FlexmonsterJSP
Version: 0.0.1-SNAPSHOT
Package: com.flexmonster.FlexmonsterJSP

6. If you see error “The superclass javax.servlet.http.HttpServlet was not found on the Java Build Path index.jsp /Flexmonster/src/main/webapp” then add Apache Tomcat to your Targeted Runtimes.


FlexmonsterJSP
Deploy
Java Re
JavaScri
Deploy
src
main
re
w
target
pom.xm
Servers
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Refactor Alt+Shift+T
Import
Export
Refresh F5
Close Project
Close Unrelated Projects
Validate
Show in Remote Systems view
Run As
Debug As
Profile As
Restore from Local History...
Maven
Java EE Tools
Team
Compare With
Configure
Source
Properties Alt+Enter


Properties for FlexmonsterJSP

Targeted Runtimes

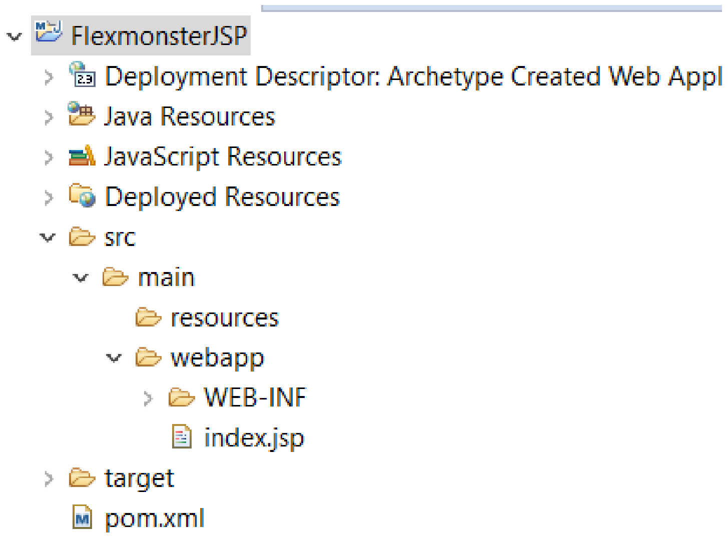
☑ Apache Tomcat v8.5

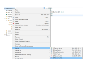
☐ Show all runtimes

Runtime composition:
<no runtime selected>

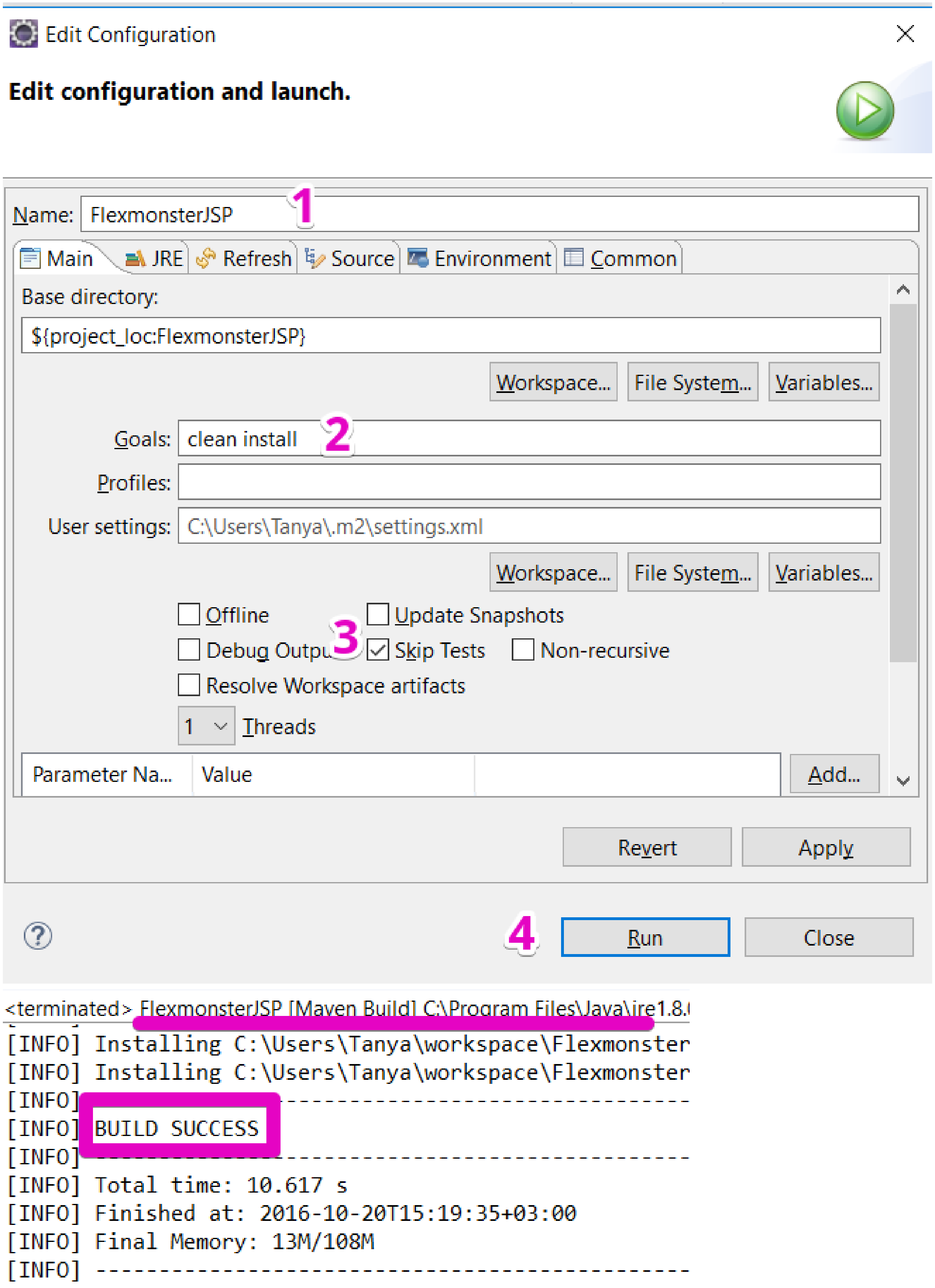
If a runtime that you want to select is not displayed or is disabled you may need to uninstall one or more of the currently installed project facets.

7. Your Maven Project should look like this.


```
FlexmonsterJSP
  Deployment Descriptor: Archetype Created Web Appl
  Java Resources
  JavaScript Resources
  Deployed Resources
  src
    main
      resources
      webapp
        WEB-INF
        index.jsp
  target
  pom.xml
```



8. Now build project with “Maven Clean Install” to check there isn’t any dependency issue with project.



Name: FlexmonsterJSP

Base directory: ${project_loc:FlexmonsterJSP}

Goals: clean install

Profiles:

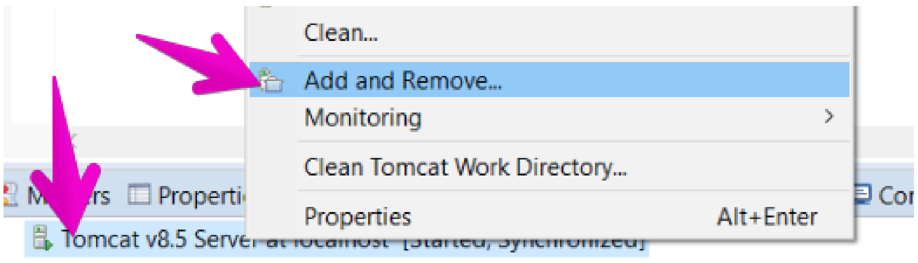
User settings: C:\Users\Tanya\.m2\settings.xml

[ ] Offline [ ] Update Snapshots
[ ] Debug Output [✓] Skip Tests [ ] Non-recursive
[ ] Resolve Workspace artifacts
1 Threads

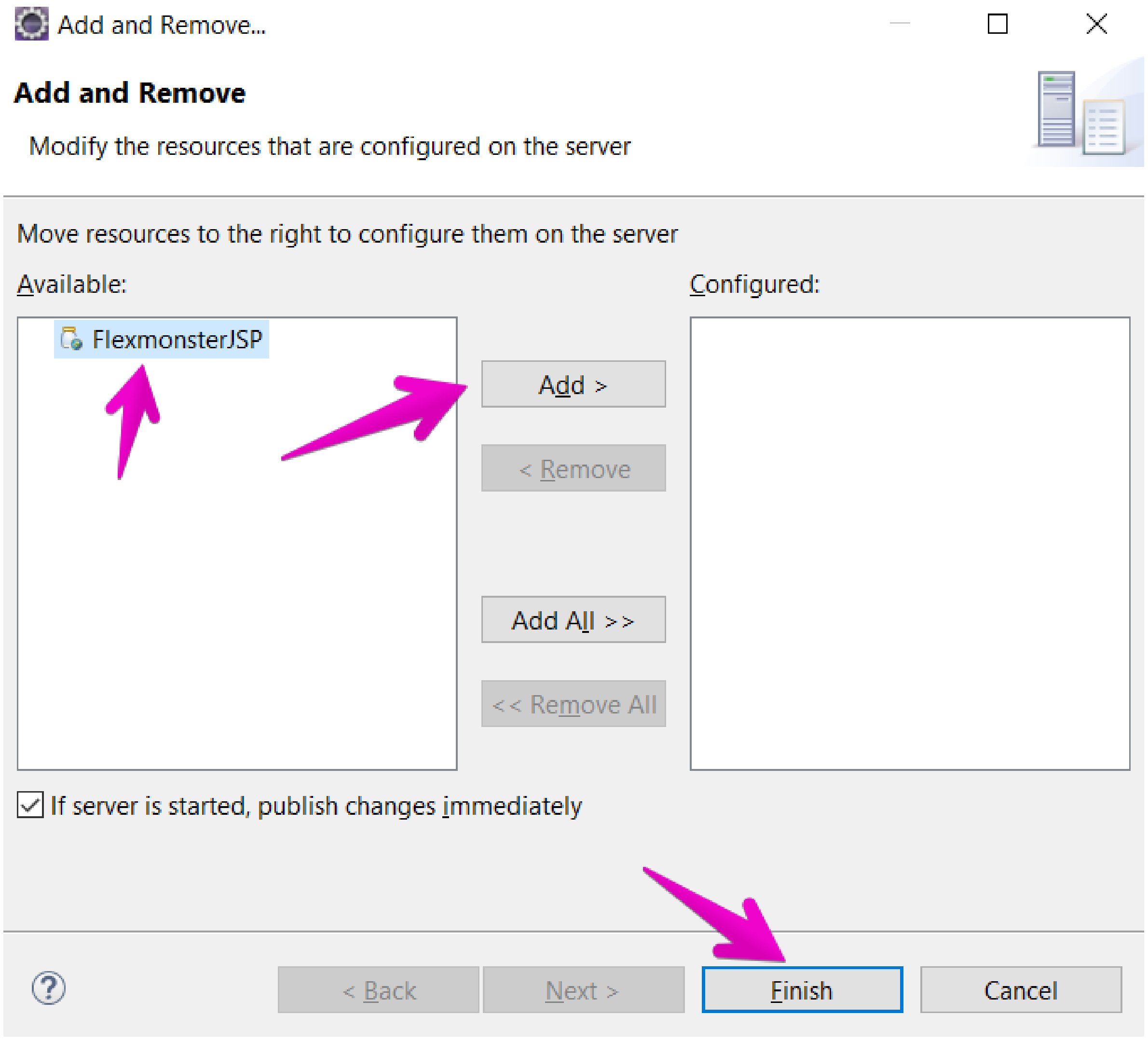
Parameter Na...	Value

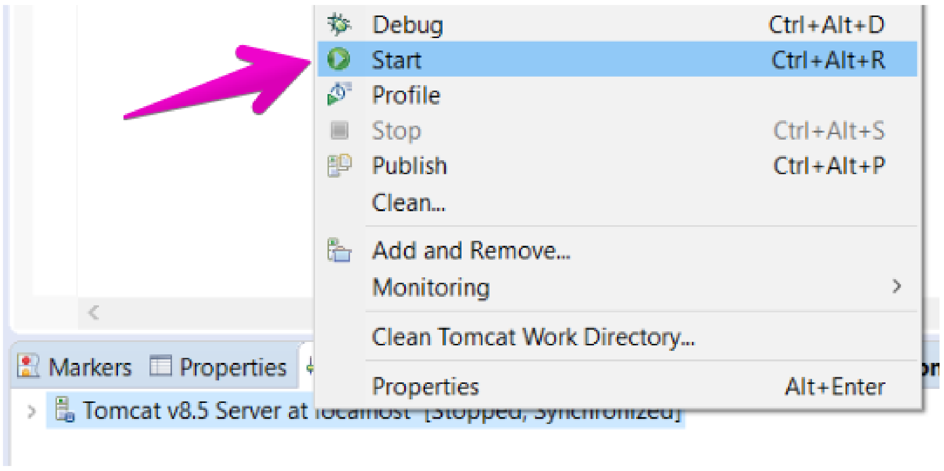
```
<terminated> FlexmonsterJSP [Maven Build] C:\Program Files\Java\jre1.8.
[INFO] Installing C:\Users\Tanya\workspace\Flexmonster
[INFO] Installing C:\Users\Tanya\workspace\Flexmonster
[INFO] ------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------
[INFO] Total time: 10.617 s
[INFO] Finished at: 2016-10-20T15:19:35+03:00
[INFO] Final Memory: 13M/108M
[INFO] ------------------------------------------------
```

9. Right click on Server -> Add and Remove.

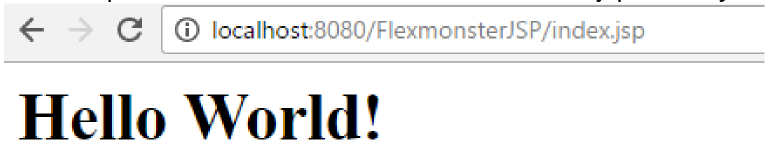

Clean...
Add and Remove...
Monitoring
Clean Tomcat Work Directory...
Properties Alt+Enter
Tomcat v8.5 Server at localhost [Started, Synchronized]

10. Select FlexmonsterJSP -> Add -> Click Finish.


Add and Remove...
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
FlexmonsterJSP
Add >
< Remove
Add All >>
<< Remove All
If server is started, publish changes immediately
< Back
Next >
Finish
Cancel



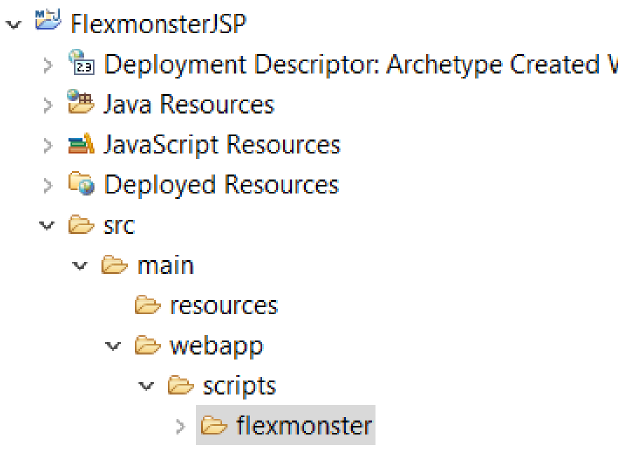
11. Start Tomcat server.
12. Visit http://localhost:8080/FlexmonsterJSP/index.jsp to see your result.



Hello World!

Step 2. Copy Flexmonster scripts

Copy flexmonster/ from the download package to src/main/webapp/scripts/ folder.



Step 3. Add Flexmonster taglib

1. Copy flexmonster-taglib.jar(https://github.com/flexmonster/pivot-jsp/blob/master/dist/flexmonster-taglib-2.3.jar)
   to src/main/webapp/WEB-INF/lib/ folder.
2. Copy flexmonster.tld(https://github.com/flexmonster/pivot-jsp/blob/master/dist/flexmonster.tld)
   to src/main/webapp/WEB-INF/lib/ folder.
3. Add the following code to src/main/webapp/WEB-INF/web.xml inside <web-app> node:

```
<taglib>
    <taglib-uri>http://flexmonster.com/taglibs</taglib-uri>
    <taglib-location>/WEB-INF/lib/flexmonster.tld</taglib-location>
</taglib>
```

Step 4. Add Flexmonster dependencies to index.jsp

- At the top add:

```
<%@ taglib prefix="flexmonster" uri="http://flexmonster.com/taglibs"%>
```

- Add jQuery and Flexmonster libraries:

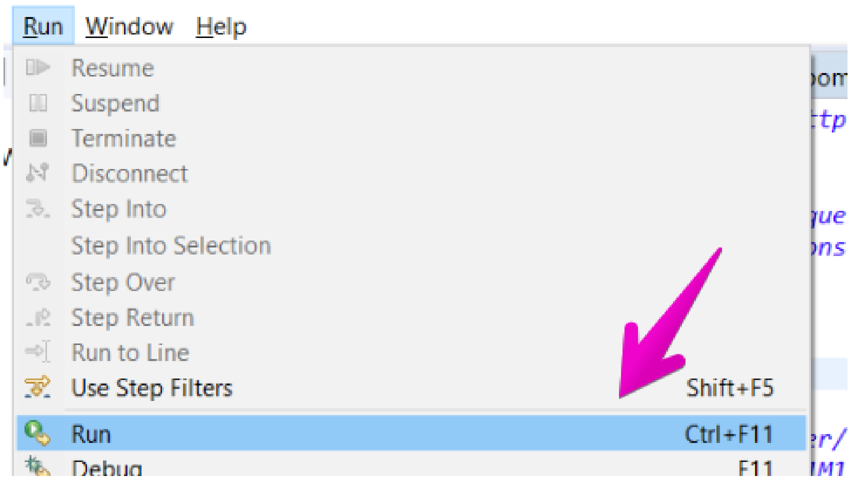
```
<script src="scripts/flexmonster/lib/jquery.min.js"></script>
<script src="scripts/flexmonster/flexmonster.js"></script>
```

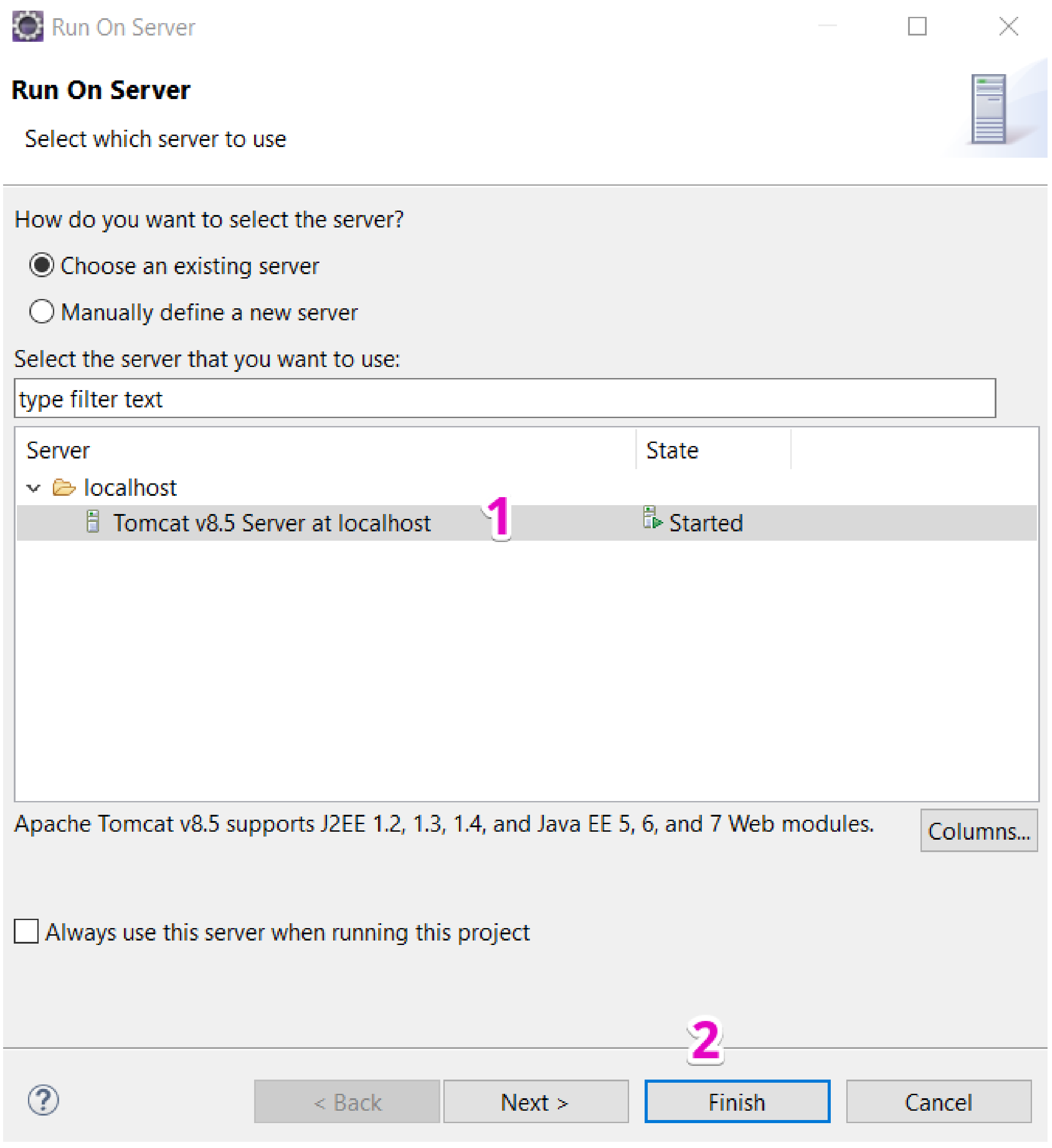
Step 5. Create Pivot Table inside index.jsp

Please note, licenseKey is your license or trial key, so you should replace XXX with an actual key:

```
<flexmonster:pivot
 name="pivot"
 componentFolder="scripts/flexmonster/"
 licenseKey="XXX"
 toolbar="true"
 report="http://www.flexmonster.com/download/report.xml"
/>
```

Step 6. Build and run the project

Choose Run -> Run and follow the steps. 


Run On Server

Run On Server

Select which server to use

How do you want to select the server?

[✓ Choose an existing server]

Manually define a new server

Select the server that you want to use:

type filter text

Server	State
localhost	
Tomcat v8.5 Server at localhost	Started

Apache Tomcat v8.5 supports J2EE 1.2, 1.3, 1.4, and Java EE 5, 6, and 7 Web modules.

☐ Always use this server when running this project

You will see Pivot Table with sample report loaded.

You can download the full example here(https://github.com/flexmonster/pivot-jsp)
.

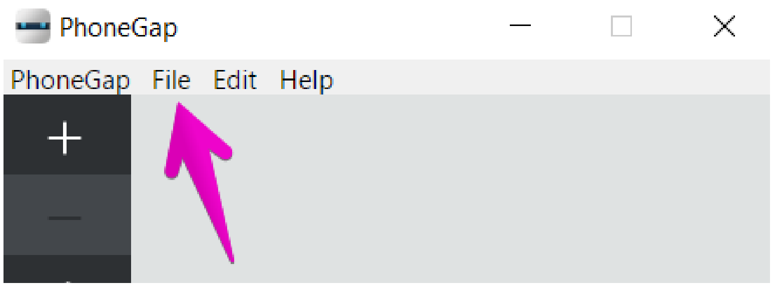
9.9. Integration with PhoneGap

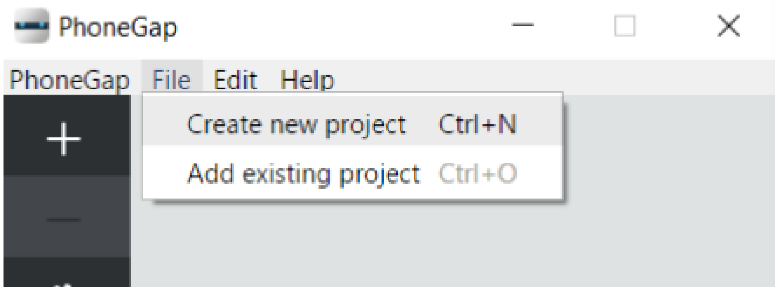
This tutorial will help you to integrate Pivot Table Component with PhoneGap(http://phonegap.com/)
. Follow the steps to setup a simple PhoneGap project with Flexmonster Pivot Table:

Step 1. Create new PnoneGap project

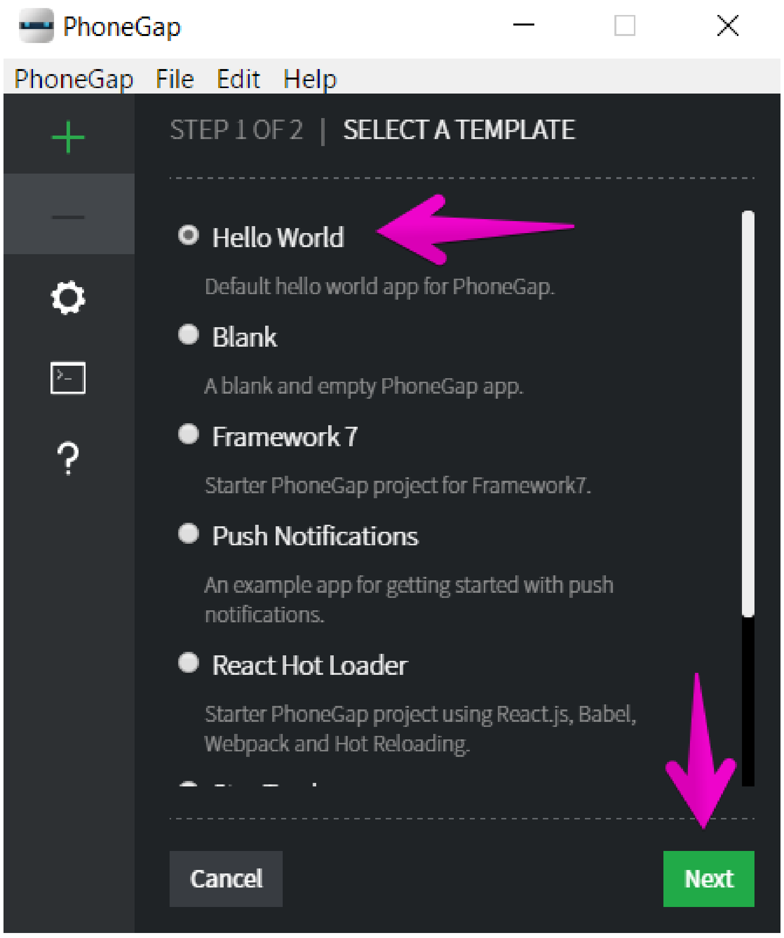
1. Open PhoneGap. If it is not installed, download and install PhoneGap using very detailed step by step

instructions here(http://docs.phonegap.com/)
.

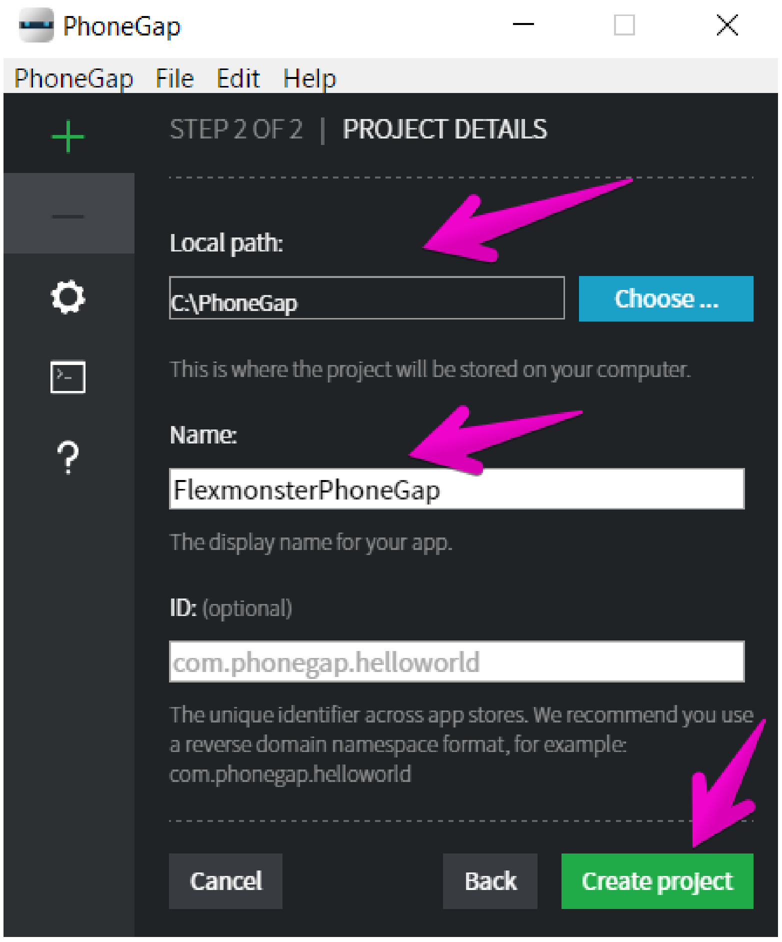
2. Click File. 

3. Choose Create new project. 

4. Choose Hello World template and click Next.


PhoneGap
PhoneGap File Edit Help
STEP 1 OF 2 | SELECT A TEMPLATE
Hello World
Default hello world app for PhoneGap.
Blank
A blank and empty PhoneGap app.
Framework 7
Starter PhoneGap project for Framework7.
Push Notifications
An example app for getting started with push notifications.
React Hot Loader
Starter PhoneGap project using React.js, Babel, Webpack and Hot Reloading.
Cancel
Next

5. Enter local path, app name and click Create project.



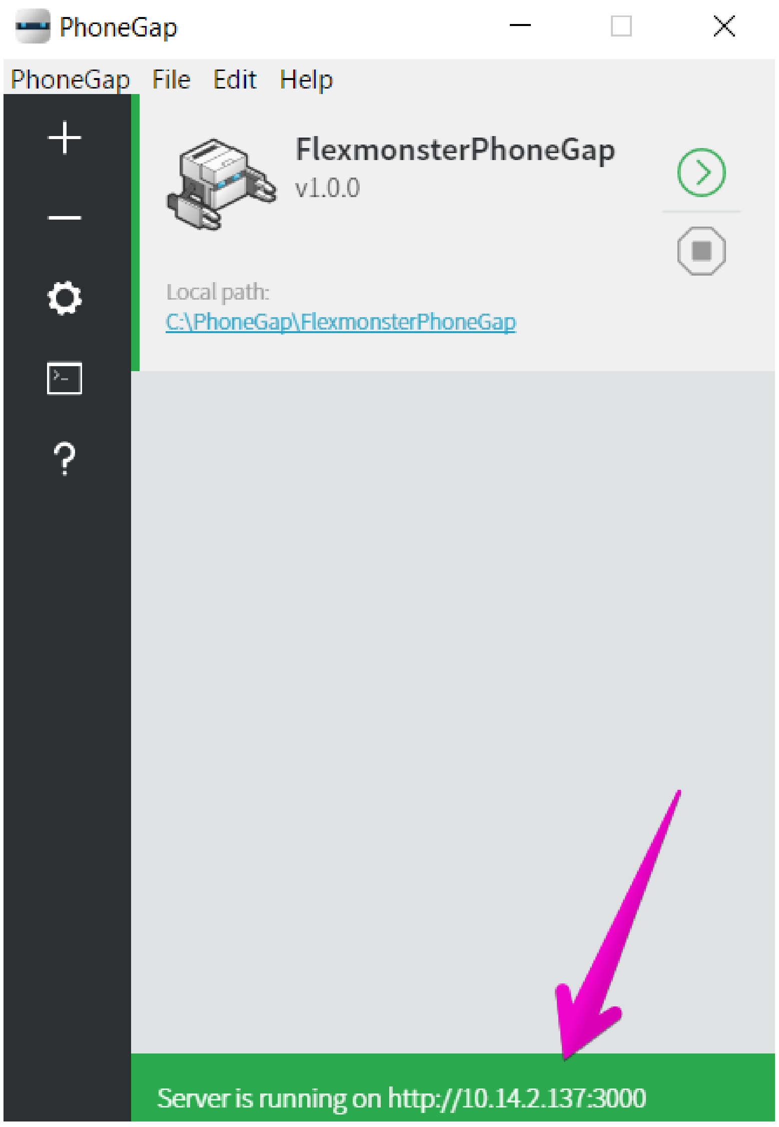
STEP 2 OF 2 | PROJECT DETAILS

Local path:
C:\PhoneGap
This is where the project will be stored on your computer.

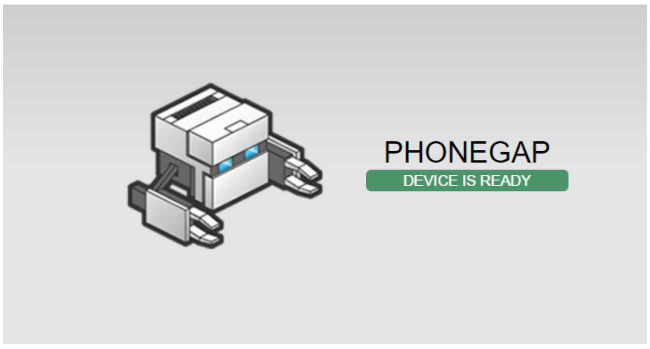
Name:
FlexmonsterPhoneGap
The display name for your app.

ID: (optional)
com.phonegap.helloworld
The unique identifier across app stores. We recommend you use a reverse domain namespace format, for example: com.phonegap.helloworld

6. Your project will run automatically. See it by clicking on the IP address at the bottom of PhoneGap

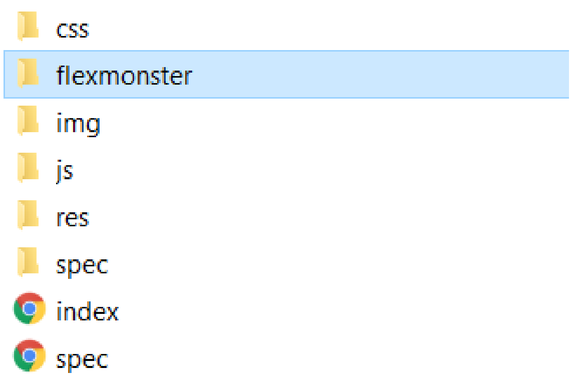
window. 

Server is running on http://10.14.2.137:3000



Step 2. Add Flexmonster files

7. Make sure you see the default page in the browser.



Copy flexmonster/ from the download package to www/ folder.

Step 3. Add Flexmonster dependencies to www/index.html

Inside www/index.html find where cordova.js is included. Add jQuery and Flexmonster libraries before cordova.js.

```
<script type="text/javascript" src="flexmonster/lib/jquery.min.js"></script>
<script type="text/javascript" src="flexmonster/flexmonster.js"></script>
<script type="text/javascript" src="cordova.js"></script>
```

Step 4. Create Pivot Table

1. Open www/index.html and create <div> container for the component inside <p class="event received"></p>. Also, remove class="blink" from <div id="deviceready" class="blink">.

   ```
   <div class="app">
     <h1>PhoneGap</h1>
       <div id="deviceready">
         <p class="event listening">Connecting to Device</p>
         <p class="event received">
           <div id="pivotContainer"
           style="position:absolute !important; left:0px; top:0px;"></div>
         </p>
       </div>
   </div>
   ```

2. Open www/js/index.js and add simple script to embed the component inside receivedEvent function. Please note, licenseKey is your license or trial key, so you should replace XXX with an actual key.

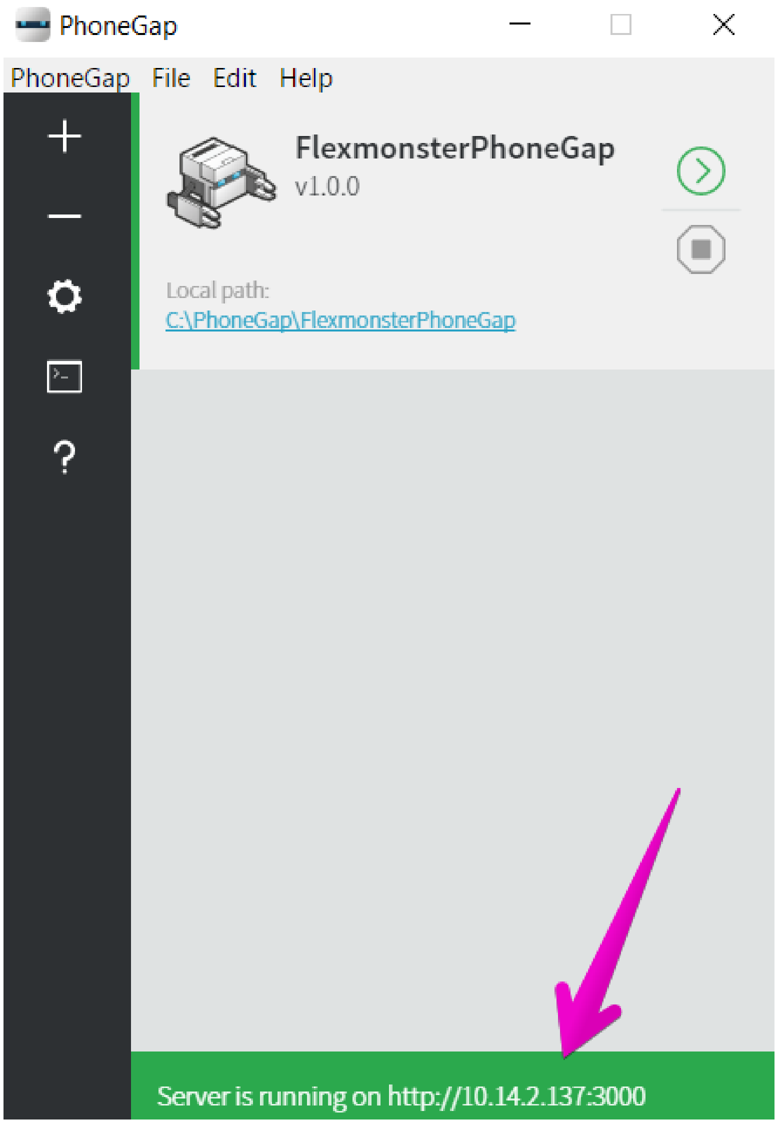
   ```
   // Update DOM on a Received Event
   receivedEvent: function(id) {
     var parentElement = document.getElementById(id);
     var listeningElement = parentElement.querySelector('.listening');
     var receivedElement = parentElement.querySelector('.received');

     listeningElement.setAttribute('style', 'display:none;');
     receivedElement.setAttribute('style', 'display:block;');

     console.log('Received Event: ' + id);
   ```

```
    var pivot = flexmonster.embedPivotComponent(
      "flexmonster/", "pivotContainer", "100%", "100%",
      {
        filename: "http://www.flexmonster.com/download/data.csv",
        licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX"
      }, true
    );
  }
```

3. See Pivot Table embedded into the project by clicking the IP address at the bottom of PhoneGap window.



Server is running on http://10.14.2.137:3000

Open www/css/index.css to configure the style of the component. For example, remove text-transform:uppercase; and letters will become normal sized.

You can download the full example here(https://github.com/flexmonster/pivot-PhoneGap)
.

10. Integration with charts

Starting from version 2.3 Pivot Table Component can be integrated with popular 3rd party visualization tools.

We added special connectors to the most popular libraries: Highcharts, FusionCharts and Google Charts.

Follow our detailed guides for each charting library:

- Highcharts(/doc/integration-with-highcharts/)
- FusionCharts(/doc/integration-with-fusioncharts/)
- Google Charts(/doc/integration-with-google-charts/)

Connection with all other 3rd party visualization tools is described in our Integration with any charting library(/doc/integration-with-any-charting-library/)
guide.

10.1. Integration with Highcharts

This tutorial will help you to integrate Pivot Component with Highcharts(http://www.highcharts.com/)
charting library.

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(http://jsfiddle.net/flexmonster/2xoqujba/)
)

- pie (demo(https://jsfiddle.net/flexmonster/j139y3xe/)
)
- polygon (demo(http://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/)
)

However, if chart type you want is not on the list it’s possible to use prepareDataFunction to preprocess the data in your own way.

Adding Highcharts

1. Add to your HTML page the following pivot table based on CSV file.

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  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="http://code.highcharts.com/highcharts.js"></script>
```

3. Add our connector for Highcharts.

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

4. Add container for the chart.

   ```
   <div id="highchartsContainer"></div>
   ```

5. Add reportComplete event handler to know when the pivot table is ready to be a data provider.

   ```
   reportcomplete: function() {
    pivot.off("reportcomplete");
    createChart();
   }
   ```

6. Add function to create the chart. This function is using a connector for Highcharts from flexmonster.highcharts.js. The connector extends Flexmonster API with the following call: highcharts.getData(options, callbackHandler, updateHandler).

   ```
   function createChart() {
    pivot.highcharts.getData(
     {},
     function(data) {
      $('#highchartsContainer').highcharts(data);
     },
     function(data) {
      $('#highchartsContainer').highcharts(data);
     }
    );
   }
   ```

You will see line chart that displays the same data that is shown in the pivot table instance pivot and it will react on updates. Check out on JSFiddle(http://jsfiddle.net/flexmonster/2xoqujba/)
.

Connector for Highcharts

We added flexmonster.highcharts.js as a connector between our component and Highcharts. 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 parameters:
  - type (optional) – String. Chart type to prepare data for. If it is not specified, data will be prepared for the line chart.

    ```
    function pie() {
     pivot.highcharts.getData({
      type: 'pie'
     }, function(data) {
    ```

```
  $('#highchartsContainer').highcharts(data);
 }, function(data) {
  $('#highchartsContainer').highcharts(data);
 });
}
```

  - slice (optional) – Object. Defines the slice for which the data should be returned (for JSON, CSV and OCSV data sources only). If not defined, the API call will return data 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(data) {
  $('#highchartsContainer').highcharts(data);
 }, function(data) {
  $('#highchartsContainer').highcharts(data);
 });
}
```

  - xAxisType (optional) – the parameter to define the type of X axis. Please use it if you want to create the chart with dates on X axis – set it to “datetime”. It can be empty or “datetime”. Try it in JSFiddle(https://jsfiddle.net/flexmonster/pzo5meke/)
    (also, this sample illustrates how to define a type for particular series).
  - valuesOnly (optional) – The default value is false. You can set it to true if all the axis in the chart should be based on numeric values. This property is applicable only for the following chart types: “bubble”, “line”, “polygon” and “spline”. Please note, that it is always true for “scatter” chart (you do not need to set it explicitly to true). Try it in JSFiddle(http://jsfiddle.net/flexmonster/wrt82p00/)
    .
  - withDrilldown (optional) – The default value is false. You can set it to true if the chart will have drilldowns. It is available for the following chart types: “area”, “areaspline”, “bar”, “column”, “waterfall”, “funnel”, “pie”, “pyramid”, “polygon”, “spline”, “line”. Try it in JSFiddle(https://jsfiddle.net/flexmonster/qf73smhf/)
    .
  - prepareDataFunction (optional) – An external function. If the connector does not include the necessary type of chart or you need to preprocess the data in a different way – please use prepareDataFunction. prepareDataFunction gets two input parameters: rawData – raw data (check the structure of rawData in getData()(52.207.238.113/api/getdata/)
    ); options – object with options set in highcharts.getData() function. Try it in JSFiddle(https://jsfiddle.net/flexmonster/x0cqafqh/)
    .
- 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. Gets two input parameters – data and rawData (rawData is passed just in case you need it, for example, for defining number formatting in the tooltip).

- updateHandler (optional) – Function. Gets two input parameters – data and rawData. Specifies what will happen once data in the Pivot Table is filtered/sorted/etc.

The connector has several functions for defining number formatting for Highcharts. All these functions get the pivot table format object and return the formatting string in Highcharts format.

- highcharts.getAxisFormat(format) – String. Gets pivot format and returns Highcharts format string for value.
- highcharts.getPointXFormat(format) – String. Gets pivot format and returns Highcharts format string for point.x.
- highcharts.getPointYFormat(format) – String. Gets pivot format and returns Highcharts format string for point.y.
- highcharts.getPointZFormat(format) – String. Gets pivot format and returns Highcharts format string for point.z.

These functions can be used in callbackHandler and updateHandler functions to define a number formatting for axes and tooltips. Try it in JSFiddle(https://jsfiddle.net/flexmonster/z0bod0w9/)
.

10.2. Integration with FusionCharts

This tutorial will help you to integrate Pivot Component with FusionCharts(http://www.fusioncharts.com/) charting library.

Supported chart types

Flexmonster supports the following FusionCharts types:

- area2d (demo(http://jsfiddle.net/flexmonster/09j1jmba/)
)
- bar2d (demo(http://jsfiddle.net/flexmonster/hwbxzmkd/)
)
- bar3d (demo(http://jsfiddle.net/flexmonster/wdLzeyfp/)
)
- column2d (demo(http://jsfiddle.net/flexmonster/2gshr9aL/)
)
- column3d (demo(http://jsfiddle.net/flexmonster/w2hhdL6a/)
)
- doughnut2d (demo(http://jsfiddle.net/flexmonster/2qdkaj4g/)
)
- doughnut3d (demo(http://jsfiddle.net/flexmonster/fzg4zuac/)
)
- line (demo(http://jsfiddle.net/flexmonster/3tu6pmh9/)
)
- marimekko (demo(http://jsfiddle.net/flexmonster/1xhod06u/)
)
- msarea (demo(http://jsfiddle.net/flexmonster/bj40dt9e/)
)
- msbar2d (demo(http://jsfiddle.net/flexmonster/vmacrbr7/)
)
- msbar3d (demo(http://jsfiddle.net/flexmonster/L1gfre6b/)
)
- mscolumn2d (demo(http://jsfiddle.net/flexmonster/heqp5me8/)
)

- mscolumn3d (demo(http://jsfiddle.net/flexmonster/ktkgxz3q/)
)
- mscolumn3dlinedy (demo(http://jsfiddle.net/flexmonster/p4fyh8sr/)
)
- msline (demo(http://jsfiddle.net/flexmonster/3tqm4gq9/)
)
- msspline (demo(http://jsfiddle.net/flexmonster/zdjgtx0j/)
)
- mssplinearea (demo(http://jsfiddle.net/flexmonster/whL2rLy1/)
)
- pareto2d (demo(http://jsfiddle.net/flexmonster/vou8zueo/)
)
- pareto3d (demo(http://jsfiddle.net/flexmonster/ej4vffo9/)
)
- pie2d (demo(http://jsfiddle.net/flexmonster/66hra3jL/)
)
- pie3d (demo(http://jsfiddle.net/flexmonster/yrm6bpw9/)
)
- radar (demo(http://jsfiddle.net/flexmonster/8g0ehgxs/)
)
- spline (demo(http://jsfiddle.net/flexmonster/5p6uL3wL/)
)
- splinearea (demo(http://jsfiddle.net/flexmonster/zuwc00oc/)
)
- stackedarea2d (demo(http://jsfiddle.net/flexmonster/sqvLps37/)
)
- stackedbar2d (demo(http://jsfiddle.net/flexmonster/6L6d4cLy/)
)
- stackedcolumn2d (demo(http://jsfiddle.net/flexmonster/qh5wpm71/)
)
- stackedcolumn3d (demo(http://jsfiddle.net/flexmonster/qjufLt2u/)
)Supported map types:
- maps/worldwithcountries (demo(http://jsfiddle.net/flexmonster/Ludwah6k/)
)

However, if chart type you want is not on the list it’s possible to use prepareDataFunction to preprocess the data in your own way.

Adding FusionCharts

1. Add to your HTML page the following pivot table based on CSV file.

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  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 our connector for FusionCharts.

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

4. Add container for the chart.

```
<div id="fusionchartContainer"></div>
```

5. Add reportComplete event handler to know when the pivot table is ready to be a data provider.

```
reportcomplete: function() {
 pivot.off("reportcomplete");
 createFusionChart();
}
```

6. Add function to create the chart. This function is using a connector for FusionCharts from flexmonster.fusioncharts.js. The connector extends 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(data) {
  chart.setJSONData(data);
  chart.render();
 }, function(data) {
  chart.setJSONData(data);
 });
}
```

You will see doughnut2d chart that displays the same data that is shown in the pivot table instance pivot and it will react on updates. Try on JSFiddle(http://jsfiddle.net/flexmonster/2qdkaj4g/)
.

Connector for FusionCharts

We added flexmonster.fusioncharts.js as a connector between our component and FusionCharts. 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 parameters:
    - type – String. Chart type to prepare data for. This property is required.
    - slice (optional) – Object. Defines the slice for which the data should be returned (for JSON, CSV and OCSV data sources only). If not defined, the API call will return data 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(data) {
    chart.setJSONData(data);
    chart.render();
  }, function(data) {
    chart.setJSONData(data);
  });
}
```

    - `prepareDataFunction (optional) - An external function. If the connector does not include the necessary type of chart or you need to preprocess the data in a different way - please use prepareDataFunction. prepareDataFunction gets two input parameters: rawData - raw data (check the structure of rawData in getData()(52.207.238.113/api/getdata/)`

```
); options - object with options set in fusioncharts.getData() function.
 Try it in JSFiddle(https://jsfiddle.net/flexmonster/sytk76tu/)
.
```

- callbackHandler - Function. Specifies what will happen once data is ready. Additional options can be speci fied and then data can be passed directly to the charting library. Gets two input parameters - data and raw Data. data is ready to be used in the chart. rawData is passed just in case you need access to rawData.m eta properties (check the structure of rawData in getData()(/api/getdata/)), for example, to define number formatting.
- updateHandler (optional) - Function. Gets two input parameters - data and rawData. Specifies what will ha ppen once data in the Pivot Table is filtered/sorted/etc.

The connector has API call for defining number formatting for FusionCharts: fusioncharts.getNumberFormat(form at:Object) - Object. Gets pivot format and returns format object for number formatting in FusionCharts. You may n eed this call when you are defining your own prepareDataFunction and want to use the number formatting from th e 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 fusioncharts.getNumberFor mat call inside prepareDataFunction:

```
var format =
    pivot.fusioncharts.getNumberFormat(data.meta.formats[0]);
for (var prop in format) {
    output.chart[prop] = format[prop];
}
```

In case you need to define a number format for the second Y axis for FusionChart, you can just add "s" prefix to each property of the returned format object when copying them to 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 in JSFiddle(https://jsfiddle.net/flexmonster/mky56ghy/)
.

10.3. Integration with Google Charts

This tutorial will help you to integrate Pivot Component with Google Charts(https://developers.google.com/chart/) charting library.

Supported chart types

Flexmonster supports the following Google Charts types:

- AreaChart (demo(https://jsfiddle.net/flexmonster/gwgueexf/)
)
- BarChart (demo(https://jsfiddle.net/flexmonster/gd5pLvmm/)
)
- ColumnChart (demo(http://jsfiddle.net/flexmonster/xdpqw7k7/)
)
- LineChart (demo(https://jsfiddle.net/flexmonster/8378ax2z/)
)
- PieChart (demo(https://jsfiddle.net/flexmonster/ygf8ekk6/)
)
- Sankey (demo(https://jsfiddle.net/flexmonster/jpy8vayk/)
)

However, if chart type you want is not on the list it’s possible to use prepareDataFunction to preprocess the data in your own way.

Adding Google Charts

1. Add to your HTML page the following pivot table based on CSV file.

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  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 our connector for Google Charts.

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

4. Add container for the chart.

   ```
   <div id="googlechartContainer"></div>
   ```

5. Add 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 function to create the chart. This function is using a connector for Google Charts from flexmonster.googlecharts.js. The connector extends 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(_data) {
 var data = google.visualization.arrayToDataTable(_data.data);
 var options = {
  title: _data.options.title,
  height: 300
 };
 var chart = new google.visualization
  .ColumnChart(document.getElementById('googlechartContainer'));
 chart.draw(data, options);
}
```

You will see column chart that displays the same data that is shown in the pivot table instance pivot and it will react on updates. Try how it works on JSFiddle(http://jsfiddle.net/flexmonster/xdpqw7k7/)
.

Connector for Google Charts

We added flexmonster.googlecharts.js as a connector between our component and Google Charts. 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 parameters:
  - type (optional) – String. Chart type to prepare data for. Possible values: "area", "bar", "column", "line", "pie", "sankey". If type is not specified, data will be prepared the same way it is prepared for "sankey".
  - slice (optional) – Object. Defines the slice for which the data should be returned (for JSON, CSV and OCSV data sources only). If not defined, the API call will return data 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 you need to preprocess the data in a different way – please use prepareDataFunction. prepareDataFunction gets two input parameters: rawData – raw data (check the structure of rawData in getData()(52.207.238.113/api/getdata/)

); options – object with options set in googlecharts.getData(). Check the example on JSFiddle(https://jsfiddle.net/flexmonster/rzrbpyc2/)
based on the data preprocessed with 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 data is ready. Additional options can be specified and then data can be passed directly to the charting library. Gets two input parameters – data and rawData (rawData is passed just in case you need it, for example, for defining number formatting in the tooltip).
- updateHandler (optional) – Function. Gets two input parameters – data and rawData. Specifies what will happen once data in the Pivot Table is filtered/sorted/etc.

The connector has several functions for defining number formatting for Google Charts. All these functions get the pivot table format object and return the formatting string in Google Charts format.

- flexmonster.googlecharts.getNumberFormat(format) – Object. Gets pivot format and returns format object for number formatting in Google Charts. This object can be used to format columns in DataTable.
- flexmonster.googlecharts.getNumberFormatPattern(format) – Object. Gets pivot format and returns Google Charts format pattern. This pattern can be used to format axes.

These functions can be used in callbackHandler and updateHandler functions to define a number formatting for axes and tooltips. Try it in JSFiddle(https://jsfiddle.net/flexmonster/dkerf354/)
.

10.4. Integration with any charting library

This tutorial will help you in the process of connecting a 3rd party visualization tool to Flexmonster Pivot Table component. Our example is simple and aims to illustrate the gist of the interaction process between data from Flexmonster and the external visualization. The example is based on d3.js(http://bl.ocks.org/mbostock/3885304)
. Integration with any other library will have the similar basic steps.

The integration is based on API call getData()(/api/getdata/)
. You should read about it to understand in which format data is returned from the component. In this article we will connect Pivot Table Component data with d3.js chart step by step:

Adding the basis for new chart

1. Add to your HTML page the following pivot table based on CSV file.

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

<script>
 var pivot = $("#pivotContainer").flexmonster({
  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 container for the chart.

```
<svg width="650" height="230"></svg>
```

3. Add reportComplete event handler to know when the pivot table is ready to be a data provider.

```
reportcomplete: function() {
 pivot.off("reportcomplete");
 createChart();
}
```

4. Add function to create the chart. This function is using getData(options, callbackHandler, updateHandler). Try it in JSFiddle(https://jsfiddle.net/flexmonster/ybaefLh4/)
.

```
function createChart() {
 pivot.getData(
  {
   // define your slice
  },
  drawChart,
  updateChart
 );
}
```

Preparing the data and drawing the chart

The most important part of drawing a chart is preparing the data, transforming it from the format returned by getData() API call to the format that suits the 3rd party visualization tool:

```
var data = prepareDataFunction(rawData);
```

Our example shows how to define and use a function (in our example it is prepareDataFunction) to process the

data. This function should prepare data in appropriate for charting library format. In this example(https://jsfiddle.net/flexmonster/ybaefLh4/)
prepareDataFunction iterates through data array records from rawData and discards a record containing grand total because it is not needed for the bar chart. Also the function renames rows from r0 to member and values from v0 to value. It is not required, but it makes the code more readable when referring 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

data array from rawData looks the following way:

```
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 be looking like this:

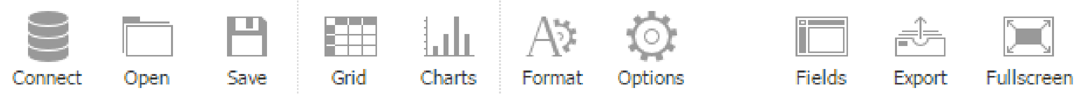
```
        {
            member:"Australia",
            value:1372281
        },
        {
            member:"France",
            value:1117794
        },
        {
            member:"Germany",
            value:1070453
        },
        {
            member:"Canada",
            value:1034112
        },
        {
            member:"United States",
            value:847331
        },
        {
            member:"United Kingdom",
            value:779899
        }
```

drawChart function draws a chart using 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)
. updateChart function works the similar way but clears SVG first.

11. Customizing toolbar

About Toolbar

Toolbar is an extra HTML/JS addition to Flexmonster Pivot Component. It uses standard API(/api/) and provides easy access to the most used features. Toolbar is free and provided “as is”.



Toolbar is available since version 2.0. Enabling toolbar is very easy, just set the appropriate parameter in $(“#pivotContainer”).flexmonster() function to true.

```
$("#pivotContainer").flexmonster({
    componentFolder:String,
    global:Object,
    width:Number,
    height:Number,
    report:Object,
    toolbar:Boolean,
    licenseKey:String})
```

Please verify that your flexmonster/ folder includes toolbar/ folder and it's not empty.

Customizing

Toolbar can be customized via beforetoolbarcreated(/api/beforetoolbarcreated/) event. Tabs and buttons can be removed from it and new ones can be added easily.

Removing tab from Toolbar

Subscribe to beforetoolbarcreated. Get all tabs using getTabs() method that returns an Array of Objects, each of them describes a tab. To remove a tab you should just delete corresponding Object from the Array. The following example will remove the first tab:

```
$("#pivot").flexmonster({
    toolbar: true,
    licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX",
    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(http://jsfiddle.net/flexmonster/x2tpc31w/)
.

Adding new tab to Toolbar

The following code will add a new tab:

```
$("#pivot").flexmonster({
    toolbar: true,
    licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX",
    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
        });
        return tabs;
    }

    var newtabHandler = function() {
     // add new functionality
    }
}
```

where:

- title – String. Label of the tab.
- id – String. Id used in CSS styles.
- handler – String. Name of the function that handles click on this tab.

Also there are another optional parameters:

- args – Any. Argument to pass to handler.
- menu – Array. Dropdown menu items.
- mobile – Boolean. If false – doesn’t show on mobile devices.
- ios – Boolean. If false – doesn’t show on iOS devices.
- android – Boolean. If false – doesn’t show on Android devices.

Check out the example with new tab on JSFiddle(http://jsfiddle.net/flexmonster/m7e74ay4/)
.

Further customization

You can customize almost everything. To explore all the options we recommend investigation of the existing code. Look at the toolbar/ folder contents (you can find it in the **[package]/**flexmonster/):

- img/ – icons and other graphic assets
- flexmonster.toolbar.js – JavaScript code
- flexmonster.toolbar.css – CSS styles

Open flexmonster.toolbar.js file. Find the Tab section (it starts with getTabs() method) to understand how it works.

To change appearance of Toolbar read Customizing appearance(/doc/customizing-appearance/)
.

12. Customizing appearance

You can customize the appearance of the Component using CSS — the same way as for the regular HTML. All CSS styles of the Component provided by Flexmonster are in flexmonster/flexmonster.css inside the package.

We recommend you writing your custom CSS in a separate file (i.e. my-flexmonster-styles.css) and keep the original styles in flexmonster.css without changes. This will help you update Flexmonster Component to the newest versions much quicker and safer. You will not break new features and will easily maintain your custom CSS.

Please check Custom CSS Demo on jsfiddle.net(https://jsfiddle.net/flexmonster/4rLgrquc/)
, which uses custom styles for the grid on top of flexmonster.css.

CSS styles for toolbar

Best practice to apply your custom styles for toolbar is to look into flexmonster/toolbar/flexmonster.toolbar.css file and create a separate CSS file on its basis. And then include it in the HTML page with Flexmonster Pivot Component. In such a way you will override styles without changing the original files.

13. Localizing component

On our website you can see localized version of Pivot Table in Demos – Localization(/demos/localization)
, for example to Spanish, Portuguese, Chinese, etc. The following article describes how Pivot Table can be localized.

Default language for all text elements in Pivot Table is English. The full process of component localization involves two steps:

1. Create localization JSON file.
2. Set localization for Pivot Table.

Step 1. Create your localization JSON file

Localization JSON file is the JSON file that has the list of all text messages and labels that are used in Pivot Table and can be localized.

If you plan to use one of localizations, that is used in our demos, you are free to download respective localization file from our site. Just click the file with language below or right click and save locally.

- English(https://github.com/flexmonster/pivot-localizations/blob/master/en.json)
- Español(https://github.com/flexmonster/pivot-localizations/blob/master/es.json)
- Français(https://github.com/flexmonster/pivot-localizations/blob/master/fr.json)
- Português(https://github.com/flexmonster/pivot-localizations/blob/master/pr.json)
- Chinese(https://github.com/flexmonster/pivot-localizations/blob/master/ch.json)
- ??????????(https://github.com/flexmonster/pivot-localizations/blob/master/ua.json)

If you have downloaded one of the above files, you can jump to Step 2. Set localization for Pivot Table(/doc/localizing-component/#SetLocalization)
.

If you use different language, you can create your own localization file. We recommend using 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.

In the following example, we localize Calculated Values window in Pivot to French.

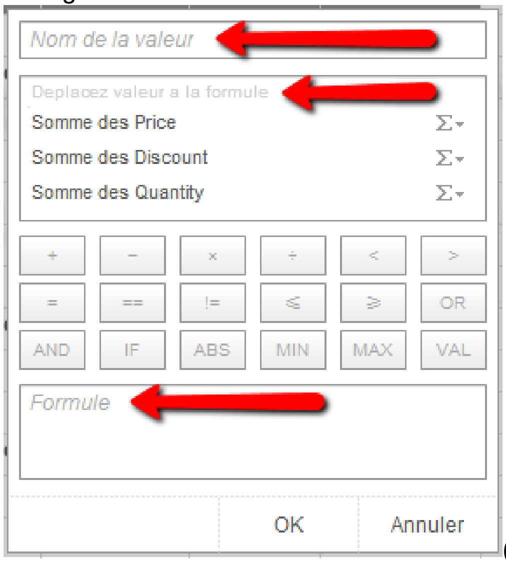
The original section of localization file looks as follows.

```
  "calculatedView": {
    "measureBox": "Drag Values To Formula",
    "measureName": "Value name",
    "formula": "Formula"
  }
```

Now we replace the respective terms with French.

```
  "calculatedView": {
    "measureBox": "Deplacez valeur a la formule",
    "measureName": "Nom de la valeur",
    "formula": "Formule"
  }
```

Once the Localization JSON file is loaded (see next step), your Pivot Table component should show the following changes in Calculated Values view



(/fm_uploads/2016/05/calculatingViewFrench.png)

Your localization can be partial. For example, if you don't need to localize Pivot Table completely, you can replace only those JSON objects that are necessary to translate. If certain label translations are not mentioned in the Localization JSON file, they will be set to default English values.

Step 2. Set localization for Pivot Table

Localization can be set as inline JSON or URL to localization JSON file. To preset localization for all reports you should set localization property in the global object. E.g. as below

URL to JSON

```
$("#pivotContainer").flexmonster({
    global: {
        localization: "loc/fr.json"
    },
    ...
});
```

JSON

```
$("#pivotContainer").flexmonster({
    global: {
        localization: {
            "calculatedView": {
                "measureBox": "Deplacez valeur a la formule",
                "measureName": "Nom de la valeur",
                "formula": "Formule"
            }
        }
    },
    ...
});
```

To specify different localization for particular report you should set localization property in the report object.

E.g. as below

URL to JSON

```
$("#pivotContainer").flexmonster({
    report: {
        localization: "loc/fr.json"
    },
    ...
});
```

JSON

```
$("#pivotContainer").flexmonster({
    report: {
        localization: {
            "calculatedView": {
                "measureBox": "Deplacez valeur a la formule",
                "measureName": "Nom de la valeur",
                "formula": "Formule"
            }
        }
    },
    ...
});
```

14. Global Object

Flexmonster component 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 reports. The details on how to configure all options in the reports are described in Configuring report article(/doc/configuring-report/)
.

It is not mandatory though to explicitly set all the options in the reports. 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 needed to modify specific default options that will be common for all the reports. This can be done using the global object. global is a parameter of flexmonster()(/api/flexmonster/)
method. In general, global object should be considered as a global parent report. It can have dataSource, options and localization sub-objects of the same structure as inside report object(/api/report-object/)
. global object overrides default option values for all the reports of the component.

Thus, in case when 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"
   }
   ```

   See the live demo on JSFiddle(http://jsfiddle.net/flexmonster/r36nyjqL/)
   .

2. The following example demonstrates how to set the read-only mode for all reports:

   ```
   global: {
     options: {
       grid: {
         showFilter: false
       },
   ```

```
    configuratorButton: false,
    sorting: false,
    drillThrough: false
  }
}
```

Check out how the example works on JSFiddle(http://jsfiddle.net/flexmonster/ge35b4fk/)
.

3. Apply date and time formatting for the component:

   ```
   global: {
     options: {
       datePattern: "'date: 'MM/dd/yy'; time: 'HH-mm"
     }
   }
   ```

   Check how it works on JSFiddle(http://jsfiddle.net/flexmonster/tqmxppj7/)
   .

4. The next example shows how you can specify dataSource for all reports:

   ```
   global: {
     dataSource: {
       filename: "http://cdn.flexmonster.com/2.3/data/data.csv"
     }
   }
   ```

   Open the example on JSFiddle(http://jsfiddle.net/flexmonster/ufv83nvf/)
   .

5. The current report is usually obtained via save() or getReport() API calls. By default, reports returned by these methods contain only options which were explicitly defined inside the report object. For example, we have the component defined the following way:

   ```
   var pivot = $("#pivotContainer").flexmonster({
     licenseKey: "XXXX-XXXX-XXXX-XXXX-XXXX",
     global: {
       localization: "loc/es.json"
     },
     report: {
       dataSource: {
        filename: "http://cdn.flexmonster.com/2.3/data/data.csv"
       }
     }
   });
   ```

save() and getReport() will get the report containing dataSource specified, but without options defined in global or defaults. Use withGlobals:true parameter in API calls to include localization, which was defined in global, to the report:

save()

```
flexmonster.save({
  withGlobals: true
});
```

getReport()

```
flexmonster.getReport({
  withGlobals: true
});
```

To include the default options to the report, use withDefaults:true parameter:

save()

```
flexmonster.save({
  withDefaults: true
});
```

getReport()

```
flexmonster.getReport({
  withDefaults: true
});
```

Test the complete example on JSFiddle(http://jsfiddle.net/flexmonster/p9keuua4/)
.

15. 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 file can be saved to the local file system or to your server.

API call exportTo()(/api/exportto/)
is used for exporting and API call print()(/api/print/)
is used for printing.

- Print(#print)
- Export to HTML(#html)
- Export to CSV(#csv)
- Export to Excel(#excel)
- Export to Image(#image)
- Export to PDF(#pdf)
- How to export on the server without using a browser(#phantomjs)

Print

Printing of the content of the grid or chart via OS print manager can be configured with the following parameters object inside print() call:

options (optional) – Object. Can contain the following print properties:

- header (starting from v2.211) (optional) – String. Header is set in 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.
- footer (starting from v2.211) (optional) – String. Footer is set in 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(http://jsfiddle.net/flexmonster/pq18xucm/)
.

Export to HTML

For exporting to HTML you should set the first parameter of exportTo() to "html". The second parameter is optional. This is the object which can contain the following properties:

- filename (optional) – the name of the resulting file. The default name is "pivotgrid".
- destinationType (optional) – defines where the component’s content will be exported. Destination type can be the following:
  - "file" (by default) – the component’s content will be exported to the file to the local computer.
  - "server" – the component’s content will be exported to the server (a server-side script is required).
- footer (starting from v2.211) (optional) – String. Footer is set in 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 (starting from v2.211) (optional) – String. Header is set in 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) – to save the file to the server you should provide the component with a path to the server-side script which can save this file.

The third parameter is optional. It is callbackHandler which gets the following object: {data: result}.

Here is how to save in HTML format as a local file:

```
var params = {
 filename: 'flexmonster'
};
flexmonster.exportTo('html', params);
```

Try the example on JSFiddle(https://jsfiddle.net/flexmonster/45e9k4c1/)
.

Saving to server:

```
var params = {
 destinationType: 'server',
 header:"<div>Table One</div>",
 footer:"<div>##CURRENT-DATE##</div>",
 url: 'your server-side script'
};
flexmonster.exportTo('html', params);
```

Export to CSV

For exporting to CSV you should set the first parameter of exportTo() to "csv". The second parameter is optional. This is the object which can contain the following properties:

- filename (optional) – the name of the resulting file. The default name is "pivot".
- destinationType (optional) – defines where the component’s content will be exported. Destination type can be the following:

  - "file" (by default) – the component’s content will be exported to the file to the local computer.
  - "server" – the component’s content will be exported to the server (a server-side script is required).
- url (optional) – to save the file to the server you should provide the component with a path to the server-side script which can save this file.

The third parameter is optional. It is callbackHandler which gets the following object: {data: result}.

Here is how to save in CSV format as a local file and get the resulting data within callbackHandler:

```
var params = {
 filename: 'flexmonster'
};
flexmonster.exportTo('csv', params,
 function(result) {console.log(result.data)}
);
```

Open the 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

For exporting to Excel you should set the first parameter of exportTo() to "excel". The second parameter is optional. This is the object which can contain the following properties:

- filename (optional) – the name of the resulting file. The default name is "pivot".
- destinationType (optional) – defines where the component’s content will be exported. Destination type can be the following:
  - "file" (by default) – the component’s content will be exported to the file to the local computer.
  - "server" – the component’s content will be exported to the server (a server-side script is required).
- excelSheetName (starting from v2.2) (optional) – String. To configure the sheet name when exporting to Excel file.
- showFilters (starting from v2.1) (optional) – Boolean. Excel only. Indicates whether the filters info will be shown (true) in exported Excel file or not (false). Default value is false.
- url (optional) – to save the file to the server you should provide the component with a path to the server-side script which can save this file.
- useOlapFormattingInExcel (starting from v2.2) (optional) – Boolean. To configure how to export grid cells in Excel file if formatting is taken from OLAP cube – as a formatted string (true) or as numbers without formatting (false). Previously it was not configurable and the cells were exported as formatted strings.

The third parameter is optional. It is callbackHandler which gets the following object: {data: result}.

Here is how to save in Excel format as a local file:

```
var params = {
 filename: 'flexmonster',
 excelSheetName: 'Report',
 showFilters: true,
 useOlapFormattingInExcel: false
};
flexmonster.exportTo('excel', 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('excel', params);
```

Export to Image

For exporting to image you should set the first parameter of exportTo() to "image". The second parameter is optional. This is the object which can contain the following properties:

- filename (optional) – the name of the resulting file. The default name is "pivotgrid".
- destinationType (optional) – defines where the component's content will be exported. Destination type can be the following:
  - "file" (by default) – the component's content will be exported to the file to the local computer.
  - "server" – the component's content will be exported to the server (a server-side script is required).
- url (optional) – to save the file to the server you should provide the component with a path to the server-side script which can save this file.

The third parameter is optional. It is callbackHandler which gets the following object: {data: result}.

Here is how to save image as a local file:

```
var params = {
 filename: 'flexmonster'
};
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

For exporting to PDF you should set the first parameter of exportTo() to "pdf". The second parameter is optional. This is the object which can contain the following properties:

- filename (optional) – the name of the resulting file. The default name is "pivot".
- destinationType (optional) – defines where the component's content will be exported. Destination type can be the following:
    - "file" (by default) – the component's content will be exported to the file to the local computer.
    - "server" – the component's content will be exported to the server (a server-side script is required).
- footer (starting from v2.211) (optional) – String. Footer is set in 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 (starting from v2.211) (optional) – String. Header is set in 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.
- pageOrientation (optional) – defines the page orientation for a PDF file. Page orientation can be the following:
    - "portrait" (by default) – defines portrait page orientation for a PDF file.
    - "landscape" – defines landscape page orientation for a PDF file.
- url (optional) – to save the file to the server you should provide the component with a path to the server-side script which can save this file.

The third parameter is optional. It is callbackHandler which gets the following object: {data: result}.

Here is how to save in PDF format as a local file:

```
var params = {
 filename: 'flexmonster',
 header:"<div>##CURRENT-DATE##</div>",
 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);
```

How to export on the server without using a browser

You can easily export on the server without using a browser by means of PhantomJS(http://phantomjs.org/)
. Please find the example on GitHub(https://github.com/flexmonster/pivot-phantomjs)
which includes:

1. Script file pivot.js for PhantomJS. Usage from the command line:

   ```
   phantomjs pivot.js
   ```

2. Pivot component itself with necessary API calls for an export generation in the index.html file.
3. Very simple save.php sample for generated files saving.

16. API reference - JavaScript(/api/)

17. API reference - Flex(/flex-api/)