Using Ranet OLAP Front-End API

Besides changing server side functionality of the solution with integrated Ranet Pivot Table, we also allow our users to change appearance and configure client side of their solutions.

There are several ways to add Pivot Table to client side application of you solution. AngularJS directive <ranetdynamicpivotgridapp ng-app="sampleDynamicPivotGrid”> is the first and the most convenient way.

These directive settings are configured through provided api. To get the access to api, you should obtain ranetAngularInstace through Angular.injector:

```
var injector = window.angular.element(document.getElementById("dynamicPivotGrid")).injector();

var api = injector.get('ranetAngularInstance');
```

After you get angular instance, you can use next api methods:

- api.addNewPage(page)
  This method adds new preconfigured page in report.
- api.deletePage(pageId, callback)
  This method removes page with given id from report. After removal callback function will be executed.
- api.setSelectedPage(pageId)
  This method sets page with given id as active.
- api.config(config)
  This method sets configuration of directive. Settings are passed via config object.

Now let’s try to configure Pivot Table via api in practice:
First of all we should configure directive itself. Let’s set connection string. In order to do this, we should pass a directive configuration object, containing connectionString field:

```
api.config({
    connectionString: "Provider=MSOLAP.4; Data Source=https://bi.galaktikasoft.com/olap/2012/msmdpump.dll; Catalog=AdventureWorksDW2012 MD-EE;"
});
```

Thereafter let’s set visibility of buttons in toolbar. To do this we should pass a directive configuration object, containing an object array, each of this objects sets visibility of specific button.

```
api.config({
    toolbar: [
        {
            name: "editConnectionButton",
            isVisible: true
        },
        {
            name: "newReportButton",
            isVisible: false
        },
        {
            name: "openReportButton",
            isVisible: false
        },
        {
            name: "saveReportButton",
            isVisible: false
        },
        {
            name: "toggleVisibilityActiveFiltersButton",
            isVisible: false
        }
    ]
});
```

You are able to configure next buttons:

- editConnectionButton
- newReportButton
- openReportButton
- saveReportButton
- toggleVisibilityActiveFiltersButton
- showCustomCalculationsEditorButton
- toggleVisibilityMetadataAndQueryDesigner
- layoutDesignerSelectorDropdown
- showCellConditionsDesignerbutton
- showExtensionSettingButton
- toggleRotateAxesButton
- pageSelectorDropdown
- showMdxQueryButton
- resetLayoutButton
- historyGroupButtons
- showSettingsButton
- exportDropdown
- importLayoutButton
- exportLayoutButton
- exportToExcelButton
- exportToXmlButton
- resetAllActionsButton

Beside properties listed earlier, there are two functions to be configured. First – importing report page from storage manager. Second – getting the identifier if terms of which your reports will be stored.

```
api.config({
    onPageImporting: function(importingPageDescritpion) {
        console.log("Now we are importing a page with id: " + importingPageDescritpion.id);
        return reportController.getModifiedImportingPage(importingPageDescritpion);
    },
    onGetFilterKey: function(currentPageDescription) {
        return reportController.getFilterKey(currentPageDescription);
    }
});
```

Now let's add page to the report. Page is set as follows:

```
var page = {
    id: "0",
    caption: "Report 1",
    cubeName: "[Adventure Works]",
    autoExecuteMdxQuery: true,
    placementLayoutModeQueryDesigner: 0,
    layout: {
        filters: [
        ],
        columns: [
            { type: "calculatedNamedSet", name: "[Set 0]" }
        ],
        rows: [
            { type: "hierarchy", uniqueName: "[Product].[Product Categories]" },
            { type: "values" }
        ],
        data: [
            { type: "measure", uniqueName: "[Measures].[Amount]" },
            { type: "calculatedMember", name: "[My custom 'Amount']" },
            { type: "kpiItem", uniqueName: "[Measures].[Financial Variance Status]" }
        ]
    },

    calculatedMembers: [
        {
            name: "[My custom 'Amount']",
            expression: "[Measures].[Amount] * 2 / 10",
            foreColor: { A: 255, R: 255, G: 255, B: 255 },
            backColor: { A: 255, R: 0, G: 150, B: 150 }
        }
    ],
    calculatedNamedSets: [
        {
            expression: "[Summary P&L]",
            name: "[Set 0]"
        }
    ],

    showMetadataAndQueryDesigner: true,
    queryDesignerSetting: {
        generateCustomCalculated: false,
        hideEmptyColumns: true,
        hideEmptyRows: false,
        includeAllMembers: false,
        includeCustomCalculationsInDrillDown: false,
        subsetCount: 0,
        useVisualTotals: true
    }
};
```

Now add configured page to the report:

```
api.addNewPage(page);
```

Beside the properties above you are also able to configure WebApi URL and path to application resources (default path is /assets/). To set these properties you should declare two variables: window.applicationUrlSpa and window.resourcePathSpa. These variables will be used afterwards while configuring AngularJS application.

```
'applicationUrl': window.applicationUrlSpa ? window.applicationUrlSpa : "",
'resourcePath': window.resoursePathSpa ? window.resoursePathSpa : "assets/",
```

Second way to integrate Pivot Table in client side application of your solution is JQuery plugin. It provides access to existing controls via JQuery object.

Some our users may have applications built with frameworks different from AngularJS. That's why besides AngularJS directive there is another realization of Ranet Pivot Table as JQuery plugin. This realization allows users to include Pivot Table regardless to the chosen architecture of application or using stack of technologies. To include this plugin in application, you should call next method on the desired JQuery object and pass Ranet Pivot Table configuration object to this method.

So let's add plugin to our page.

```
$('#dynamicPivotGridControl').ranetDynamicPivotGrid({
    connectionString: "Provider=MSOLAP.4; Data Source=https://bi.galaktikasoft.com/olap/2012/msmdpump.dll; Catalog=AdventureWorksDW2012 MD-EE;",
    locale: "en",
    idDefaultPage: "0",
    applicationUrl: "",
    templatesPath: "assets/templates/",
    pages: [
        page1,
        page2
    ]
});
```

page1 and page2 are the same pages as given above.

You can find more details about configuring JQuery plugin at the ranetDynamicPivotGrid.jquery.html page.