**Graph Demo Readme**

The Graph demo shows how to use the PowerBuilder WebBrowser control to render the third-party visual charts which are exposed as JavaScript classes. It mainly demonstrates how to execute JavaScripts in the PowerBuilder application to dynamically adjust data display in different chart styles; and connect the chart event with the WebBrowser event so that JavaScript and PowerScript can interact with each other.

**General steps to apply a chart (exposed as JavaScript classes):**

Step 1: Create an HTML page which contains the following:

* References to the online JS and CSS files for the chart.
* A div or canvas container where the chart will be rendered.
* An event listener for the Resize event, so that the div or canvas container can be automatically resized.

Step 2: Create a PowerBuilder window and add a WebBrowser control to the window. Set the WebBrowser control to open the HTML page created in step 1.

Step 3: Define a ue\_clicked user event for the WebBrowser control and register it before it’s triggered.

Step 4: Use the EvaluateJavaScriptSync and EvaluateJavaScriptAsync functions of the WebBrowser control to generate the chart.

Step 5: If the JavaScript classes support user event bindings, then bind window.webBrowser.ue\_clicked with the WebBrowser ue\_clicked event so that arguments can be passed between them.

**To apply Google Charts:**

Step 1: Create an HTML page which contains the following:

* References to the online JS and CSS files for Google Charts. Note that Internet connection is required in order to reference the online files. The referenced JS and CSS files may vary according to the chart styles.
* A div container where the chart will be rendered.
* An event listener for the Resize event, so that the div container can be automatically resized. See *window.addEventListener* in charts.html.
* A function that executes the event of the window.webBrowser object. See *function selectHandler(e)* in charts.html. webBrowser is a reserved keyword (case-sensitive) in PowerScript that binds with the PowerBuilder WebBrowser control.

For a sample HTML page, see charts.html.

Step 2: Create a PowerBuilder window and add a WebBrowser control to the window. Set the defaultUrl property of the WebBrowser control to open the HTML page created in step 1.

For example, defaultUrl=”[file:///charts.html](file://C:\charts.html)”

For a sample window and WebBrowser control, see the n\_webbrowser user object in this demo.

Step 3: Define a user event for the WebBrowser control and then register it before it’s triggered.

For example,

event ue\_clicked(string as\_arg);

end event

RegisterEvent("ue\_clicked")

For a sample code, see the n\_webbrowser user object in this demo.

Step 4: Use the EvaluateJavaScriptSync and EvaluateJavaScriptAsync functions of the WebBrowser control to execute JavaScripts to define three variables for the HTML page:

* data – the value comes from the DataWindow, and generated as JSON strings through JSONGenerator
* myChart - myChart.draw(data, options) generates the chart
* options

For a sample code, see of\_apply and of\_apply\_async functions in n\_webbrowser.

Step 5: Bind the select event of the point in the chart with the selectHandler function, so that the WebBrowser ue\_clicked event can be triggered when window.webBrowser.ue\_clicked is triggered, and values can be passed by as\_arg between PowerScript and JavaScript.

**To import Google Charts from demo**

The demo app has shown cases for many chart styles including Line, Candlestick(K-Line), Area, Column, Pie, Scatter, Org, and Timelines. Notice that, for Org and Timelines charts, additional JavaScript and CSS files must be referenced.

If you want to directly integrate the Google Charts from the demo app into your app, take the following steps:

* Add googlecharts\_base.pbl to your application.
* Call the interface in the n\_webbrowser user object of googlecharts\_base.pbl.

n\_webbrowser has encapsulated the interfaces required for interfering with the chart.

The of\_createData function etc. generate the JSON string from the graph DataObject, DataWindow, and SQL statements and then generate the chart from the JSON string. For example,

String ls\_Title, ls\_Option, ls\_data

//Title

ls\_Title = dw\_1.Describe("gr\_1.title")

wb\_1.of\_SetTitle(ls\_Title)

//Style

wb\_1.of\_SetStyle("area")

//Width

wb\_1.of\_SetWidth(750)

//Height

wb\_1.of\_SetHeight(400)

//CreateData

ls\_data = wb\_1.of\_createdata\_graph(dw\_1)

wb\_1.of\_SetData(ls\_data)

//CreateOption

ls\_Option = wb\_1.of\_createoption()

//SetOption

wb\_1.of\_SetOption(ls\_Option)

//Apply

wb\_1.of\_Apply()

The other options for the chart are configured through the options variable. If not configured, the chart will use the default system settings.

For example, to configure the color, you can add an array in the options argument:

{title:'Title', legend:{position:'bottom'},colors:['red','#004411','rgb(128,128,128)']}

**To apply ECharts:**

Step 1: Create an HTML page which contains the following:

* References to the offline JS files for ECharts: echarts.js and echarts-gl.js.
* A div container where the chart will be rendered.
* An event listener for the Resize event, so that the div container can be automatically resized.

For a sample HTML page, see echarts.html.

Step 2: Create a PowerBuilder window and add a WebBrowser control to the window. Set the defaultUrl property of the WebBrowser control to open the HTML page created in step 1.

For example, defaultUrl=”file:///echarts.html”

For a sample window and WebBrowser control, see the n\_webbrowser user object in this demo.

Step 3: Define a user event for the WebBrowser control and then register it before it’s triggered.

For example,

event ue\_clicked(string as\_arg);

end event

RegisterEvent("ue\_clicked");

For a sample code, see the n\_webbrowser user object in this demo.

Step 4: Use the EvaluateJavaScriptSync and EvaluateJavaScriptAsync functions of the WebBrowser control to execute JavaScripts to define two variables for the HTML page:

* options - the value comes from the DataWindow, and generated as JSON strings through JSONGenerator
* myChart - myChart.setOption(option) generates the chart

For a sample code, see of\_apply and of\_apply\_async functions in n\_webbrowser.

Step 5: Bind the select event of the point in the chart with the selectHandler function, so that the WebBrowser ue\_clicked event can be triggered when window.webBrowser.ue\_clicked is triggered, and values can be passed by as\_arg between PowerScript and JavaScript.

**To import ECharts from demo**

The demo app has shown cases for many chart styles including Line, Candlestick(K-Line), Area, Bar, Pie, Scatter, Radar, Funnel and Gauge.

If you want to directly integrate the ECharts from the demo app into your app, take the following steps:

* Add echarts\_base.pbl to your application.
* Call the interface in the n\_webbrowser user object of echarts\_base.pbl.

n\_webbrowser has encapsulated the interfaces required for interfering with the chart.

The of\_CreateOption function etc. generate the JSON string from the graph DataObject, DataWindow, and SQL statements and then generate the chart from the JSON string. For example,

String ls\_Title, ls\_Option, ls\_data

Boolean lb\_Toolbox

//Title

ls\_Title = dw\_1.Describe("gr\_1.title")

wb\_1.of\_SetTitle(ls\_Title)

//Style

wb\_1.of\_SetStyle("area")

//Width

wb\_1.of\_SetWidth(750)

//Height

wb\_1.of\_SetHeight(400)

//ToolBox

wb\_1.of\_SetToolBox(lb\_Toolbox)

//CreateOption

ls\_Option = wb\_1.of\_CreateOption\_Graph(dw\_1)

//SetOption

wb\_1.of\_SetOption(ls\_Option)

//Apply

wb\_1.of\_Apply()

The other options for the chart are configured through the options variable. If not configured, the chart will use the default system settings.

For example, to configure the color, you can add an array in the options argument:

{title:{text:'default title',left:'center'},color:['red','#004411','rgb(128,128,128)']}

Note: The current version of echarts.js is 5.1.2 and echarts-gl.js is 2.0.6. You can download and use a newer version of these files from <https://github.com/apache/echarts/releases> and https://github.com/ecomfe/echarts-gl/tree/master/dist.

**Confirming or adjusting the default PowerClient and PowerServer project settings:**

* The demo application contains default PowerClient and PowerServer projects. Please double check the project settings against your environment, to make sure the projects can work correctly.

**With both the PowerClient and PowerServer projects:**

* Setting up the local web server. The demo project is deployed to local web server by default. If you go with the local server, make sure:

1. IIS is running on the current machine:
2. The “local” web server profile is configured in the Web Server Profile configuration, and Cloud App Launcher is uploaded to the local server.

**With the PowerServer projects:**

* License settings in the Web APIs tab

Please import a valid license into the project settings using Auto Import (importing the current PowerBuilder CloudPro or trial license), or Import from File (file from the License Management page on https://account.appeon.com).

* Solution location in the Web APIs tab

The default location is set to [current user]\source\repos. If it does not exist on the current machine, please select a valid one.

* Web API URL in the Web APIs tab

Make sure the port setting in the Web API URL is not occupied by another program. Also, if you plan to apply a web debugging proxy tool to debug the deployed application, avoid using “localhost” in the URL. Instead, change to the actual IP address.

* Database Configuration in the Web APIs tab

If you use SQL Anywhere as the demo database, no change is needed to the database configuration.

If you use PostgreSQL as the demo database, the default login account is postgres (user)/postgres (password). Please double check the connection.