Enterprise Web Developer

Using the Emprise Javascript Charting Widgets

Version 4.0.682.2

Background

As of build 4.0.682, Enterprise Web Developer (EWD) includes advanced functionality that allows you to make use of the uniquely powerful Javascript charting widget libraries from Emprise Javascript Charts (EJSChart) (http://www.ejschart.com). By incorporating these widgets into your applications, you'll be able to quickly and easily add sophisticated graphs and charts to your web applications, and without the need for any browser plug-ins or the use of SVG! Unlike users of these widget libraries who are forced to hand-craft the required code, you'll be able to let EWD do all the hard work. With EWD you'll be creating in minutes what others will take days or weeks to accomplish. Your web applications will never look the same again!

This document is a reference guide to using EWD's EJSChart custom tags.

Pre-requisites

This guide assumes that you installed EWD (build 682 or later) and have read the eXtc Web Developer Overview, the Installation Guide, Tutorial and Ajax documents. You should also have the main EWD API Guide at hand.

You'll also need to install the EJSChart Javascript library files. Full instructions are available at their web site:

- ExtJS: http://www.ejschart.com

Please note the licensing requirements for Emprise Javascript Charts: this is a commercial product and not free of charge. It is your responsibility to comply with these licensing requirements. Note that you cannot use the free Personal version if you want to use charts within EWD Ajax applications: doing so will result in Javascript errors.

EWD's Emprise Javascript Charts Custom Tags

EWD provides a set of custom tags that represent the various EJSChart widgets. The ExtJS custom tags have a prefix of *ejsc:* For example:

```
<ejsc:chart>
<ejsc:lineSeries>
```

EWD's compiler will convert these into the corresponding Javascript and HTML tags that you would otherwise have to write by hand. This approach provides you with a much simpler and intuitive way of using EJSChart widgets, some of which can be very complex and require a significant amount of advanced Javascript programming.

EWD also automatically handles all the many additional complexities that can occur when using EJSChart widgets within an Ajax application, where whole sections of markup containing one set of widgets are replaced with new markup containing new sets of widgets. With EWD you can focus on ***what*** you're wanting to do with the EJSChart widgets, not ***how*** they actually work.

The Build 682 of EWD implements the full set of widgets in EJSCharts Version 1.2.1.

EJSChart widgets are highly configurable via their *properties*. EWD allows you to apply these properties by mapping them to correspondingly named attributes within each EJSChart custom tag. Usually EWD will automatically apply a set of sensible defaults for properties that you don't specify. Values for all config options can be either string literal values or session variables, for example:

```
<ejsc:lineSeries id="chart1Series1" datastore="myArr" color="#myColor" />
```

*The color value will be determined by a session variable named **myColor**, denoted by the # character, whilst the id and datastore values use fixed literal values.*

This document will not list the various config options for each EJSChart custom tag. If you wish to find out the config options that are available for each custom tag, then you should refer to the EJSChart API documentation at http://www.ejschart.com/help/.

EWD Configuration

When you install the EJSChart libraries, you'll usually do so by creating a sub-directory under your web server's root path into which all the EJSChart resurce files are copied, eg:

*C:\Program Files\Apache Group\Apache2\htdocs**EJSChart***

or

*/var/www/html/**EJSChart***

If so, the various EJSChart Javascript files will be referred to, within your web pages, via a relative URL based on the sub-directory name you chose, eg:

/EJSChart

EWD provides a simple shortcut for loading the EJSChart Javascript files into your applications' pages. After the initial <ewd:config> tag at the top of your EWD pages, simply add:

*<ejsc:config path="**/EJSChart**" />*

Substitute the path as appropriate to your EJSChart installation.

If you are developing an EWD Ajax application, you'll need this EJSChart configuration tag in only your container page(s). It should *not* appear in your EWD page fragments.

EJSChart Charts: Basics

There are two parts to an EJSChart graphic:

- the *chart* container, which defines the styling and properties of the overall container for a specific chart, eg its axes, legends, size, colour scheme and other styling properties;
- one or more *series* that are plotted within the chart container. EJSChart provides the following types of series:

    - line series — line plot graphs
    - area series — as above, but the area under the plot filled in
    - scatter series — unconnected plotted points
    - bar series — histogram
    - pie series — pie charts
    - analog guage series — an analog dial

The data for each series is held in a *datastore*. This is a JSON-based data structure which is delivered to the browser from the back-end system. EWD looks after all the JSON-specific aspects of the datastore creation and transmission, leaving you just the simple task of specifying the array of data that will populate the datastore and hence the series.

Provided you have anything other than the free Personal version of EJSChart, you can plot as many graphs, containing as many series (of the same or different types) within your web pages.

EWD also looks after the mechanics of using EJSChart widgets within Ajax page fragments.

A Simple Line Series

Let's start by creating a simple line chart in a standard EWD web page.

First, create a new EWD application sub-directory named *ejscDemo*. Eg if your EWD Application Root Path is c:\ewdApps, create:

c:\ewdApps\ejscDemo

Copy the page definition below and save it in a file named chart*Demo1.ewd* in the *ejscDemo* application directory.

```
<ewd:config isFirstPage="true" />
<ejsc:config path="/EJSChart" />

<html>
  <head>
    <title>EWD/EJSChart Demo</title>
  </head>
  <body>

<ejsc:chart id="chart1" width="400px" height="300px" show_legend="false" title="Line Graph"
x_axis_caption="Day" y_axis_caption="Quantity">
  <ejsc:lineSeries id="chart1Series1" datastore="myArr" color="rgb(0,0,250)" />
</ejsc:chart>

  </body>
</html>
```

Note: EWD does not currently allow tags to be split across more than one line. The example boxes in this document will show some lines wrapped across 2 or more lines. Make sure you reduce these to a single line per tag in your EWD source files.

You should then have a file:

c:\ewdApps\ejscDemo\chartDemo1.ewd

Compile this page to the technology of your choice, eg:

d compileAll^%zewdAPI("ejscDemo",,"php")

The first time you compile this page you should compile the complete application to ensure that EWD generates all the additional supporting pages. Thereafter you can just recompile the page on its own using:

d compilePage^%zewdAPI("ejscDemo","chartDemo1",,"php")

If you try to run the compiled version of the *chartDemo1* page, you'll find that you get an *Invalid Request* error. This is because we haven't created the datastore named *myArr* that is referred to in the <*ejsc:lineSeries*> tag.

Datastores are the responsibility of the programmer to create, and this is done in the *prePageScript*. Add a pre-page script to the <*ewd:config*> tag:

```
<ewd:config isFirstPage="true" prepagescript="##class(ewd.test).getDemoChart1Data">
```

In the example above, we're going to call a Caché class method named *getDemoChart1Data* in a class named *ewd.test*. You can change this to use a class of your own choice. Here's the source code for the class method:

```
ClassMethod getDemoChart1Data(sessid As %String) As %String
{
f i=1:1:50 s data(i)=$r(20000)
d createDataStore^%zewdEJSC(.data,"myArr",sessid)
QUIT ""
}
```

In this example we're simply hard-coding the creation of the values we wish to plot: in this case 50 random numbers between 0 and 19,999. Of course, in real-world examples the values would be fetched or derived from your database. However, the pattern will be the same when creating the datastore for most series:

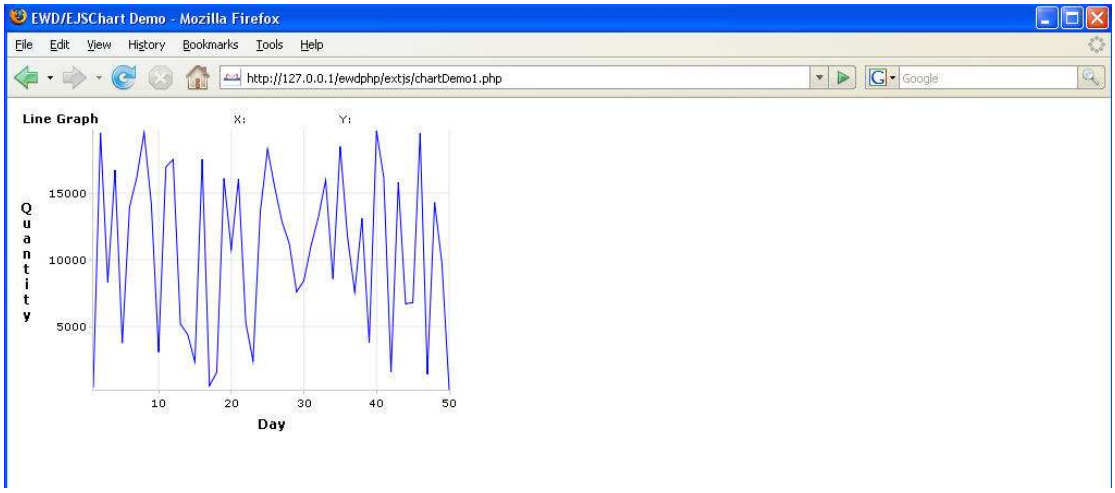
- create a 1-dimensional local array: *array(x-coordinate) = y-coordinate*
- pass this array into the createDatastore function that EWD provides for you in the %zewdEJSC routine.

As you can see, the programmer's task is straightforward and intuitive. He/she needs to know nothing about how the EJSChart datastore is structured in JSON terms, nor how it is conveyed to the browser.

Equally, the page designer's task is straightforward and intuitive. Specify the series and its properties, and specify the name of the datastore that will populate it.

Compile and save the class method you created above.

If you've installed and configured everything properly, when you run the compiled *chart1Demo* page in a browser, you should see something like the following:



Your first EJSChart graph, in minutes!

Let's take a closer look at the EJSChart tags you used in this example:

```
<ejsc:chart id="chart1" width="400px" height="300px" show_legend="false" title="Line Graph"
x_axis_caption="Day" y_axis_caption="Quantity">
  <ejsc:lineSeries id="chart1Series1" datastore="myArr" color="rgb(0,0,250)" />
</ejsc:chart>
```

The <ejsc:chart> tag defined the following:

- id — optional id for the chart. If you leave this out, EWD's compiler will automatically assign an id
- width — the width of the area assigned to the chart
- height — the height of the area assigned to the chart
- show_legend — whether or not to add a legend to the chart
- title — the title to display on the chart
- x_axis_caption — The caption to display below the x-axis
- y_axis_caption — The caption to display beside the y-axis

Try modifying these properties and see the effects on the graph. Take a look at the EJSChart API documentation and you'll see other properties that you can use. In particular, you may want to use the legend and turn off the mouse interactivity. For example:

```
<ejsc:chart id="chart1" width="500px" height="400px" show_legend="true"
allow_interactivity="false" legendTitle="Random numbers" show_mouse_position="false"
title="Line Graph" x_axis_caption="Day" y_axis_caption="Quantity">
```

The <ejsc:lineSeries> tag defined the following:

- id — optional id for the series. If you leave this out, EWD's compiler will automatically assign an id
- datastore — the name of the JSON datastore that holds the series' data
- color — the colour of the line used to connect the series points.

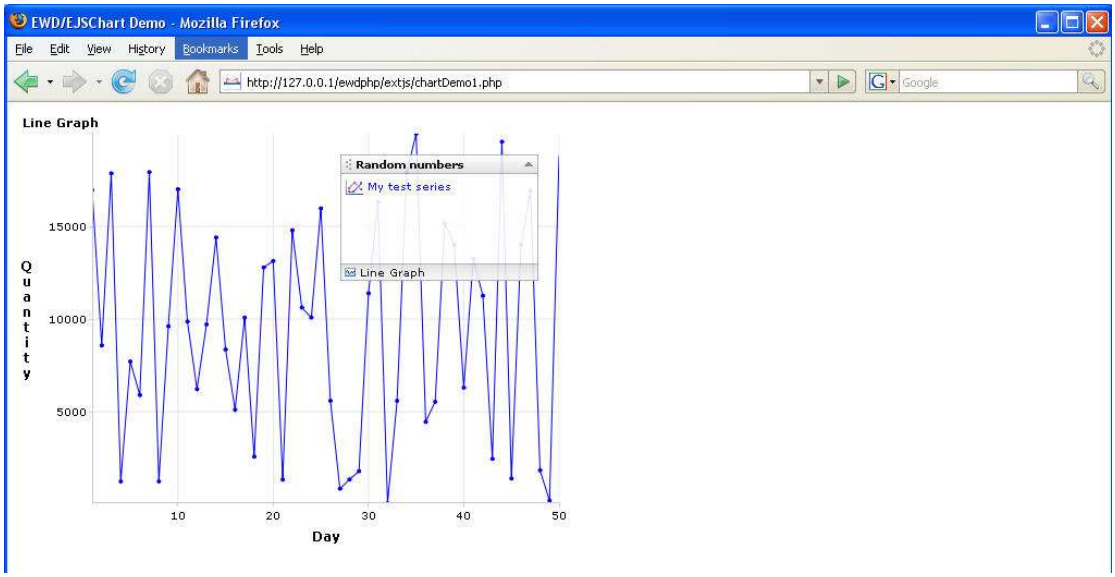
Try modifying these properties and see the effects on the graph. Take a look at the EJSChart API documentation and you'll see other properties that you can use. For example you can specify your own legend title for this series and display the data points:

```
<ejsc:lineSeries id="chart1Series1" datastore="myArr" drawPoints="true" title="My test series"
color="rgb(0,0,250)" />
```

So if you combine these changes:

```
<ejsc:chart id="chart1" width="500px" height="400px" show_legend="true"
allow_interactivity="false" legendTitle="Random numbers" show_mouse_position="false"
title="Line Graph" x_axis_caption="Day" y_axis_caption="Quantity">
 <ejsc:lineSeries id="chart1Series1" datastore="myArr" drawPoints="true" title="My test series"
color="rgb(0,0,250)" />
</ejsc:chart>
```

then recompile the page and refresh the browser, you should now see something like:



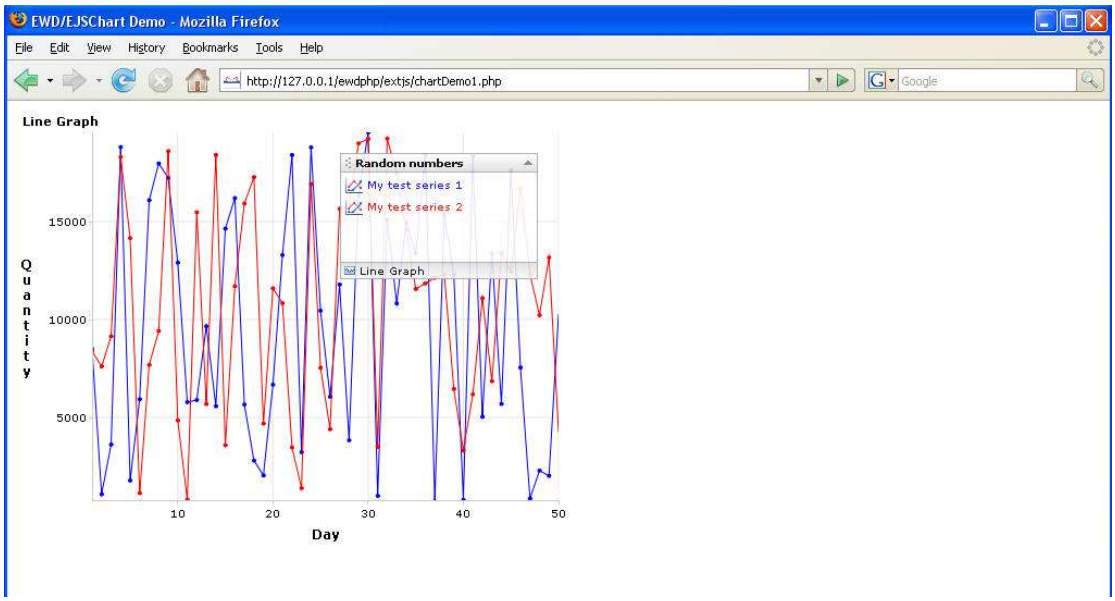
Now let's try adding a second line series to the graph. First change the page as follows:

```
<ewd:config isFirstPage="true" prepagescript="##class(ewd.test).getDemoChart1Data"/>
<ejsc:config path="/EJSChart" />

<html>
  <head>
    <title>EWD/EJSChart Demo</title>
  </head>
  <body>

<ejsc:chart id="chart1" width="500px" height="400px" show_legend="true"
allow_interactivity="false" legendTitle="Random numbers" show_mouse_position="false"
title="Line Graph" x_axis_caption="Day" y_axis_caption="Quantity">
 <ejsc:lineSeries id="chart1Series1" datastore="myArr1" drawPoints="true" title="My test series
1" color="rgb(0,0,250)" />
 <ejsc:lineSeries id="chart1Series2" datastore="myArr2" drawPoints="true" title="My test series
2" color="rgb(250,0,0)" />
</ejsc:chart>

  </body>
</html>
```

and then modify the prePageScript to create both datastores:

```
ClassMethod getDemoChart1Data(sessid As %String) As %String
{
f i=1:1:50 s data(i)=$r(20000)
d createDataStore^%zewdEJSC(.data,"myArr1",sessid)
f i=1:1:50 s data(i)=$r(20000)
d createDataStore^%zewdEJSC(.data,"myArr2",sessid)
QUIT ""
}
```

Save and compile the class method and recompile the chartDemo1.ewd page. When you refresh the browser it should now look something like this:

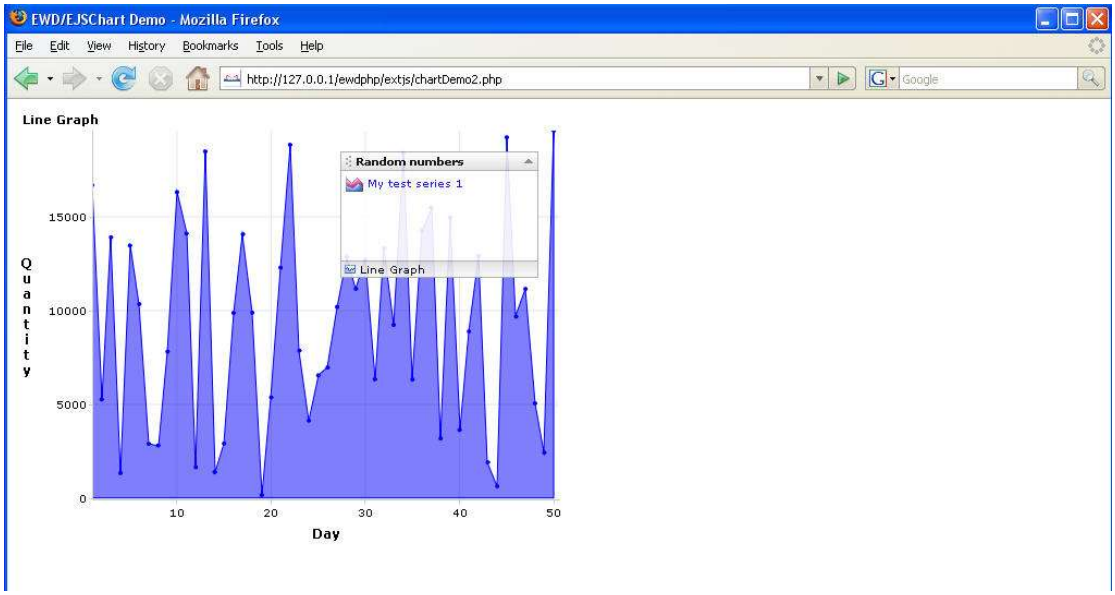


Area Series

The area series is very similar to the line series and uses exactly the same datastore. So we can just adapt the *chartDemo1.ewd* page to create *chartDemo2.ewd*:

```
<ewd:config isFirstPage="true" prepagescript="##class(ewd.test).getDemoChart1Data"/>
<ejsc:config path="/EJSChart" />

<html>
  <head>
    <title>EWD/EJSChart Demo</title>
  </head>
  <body>

<ejsc:chart id="chart1" width="500px" height="400px" show_legend="true"
allow_interactivity="false" legendTitle="Random numbers" show_mouse_position="false"
title="Line Graph" x_axis_caption="Day" y_axis_caption="Quantity">
 <ejsc:areaSeries id="chart1Series1" datastore="myArr1" drawPoints="true" title="My test series
1" color="rgb(0,0,250)" />
</ejsc:chart>

  </body>
</html>
```

We can use the same prePageScript (although we'll only be using the first datastore it creates). So save the new page, compile it and load the compiled version into your browser. You should now see something like this:



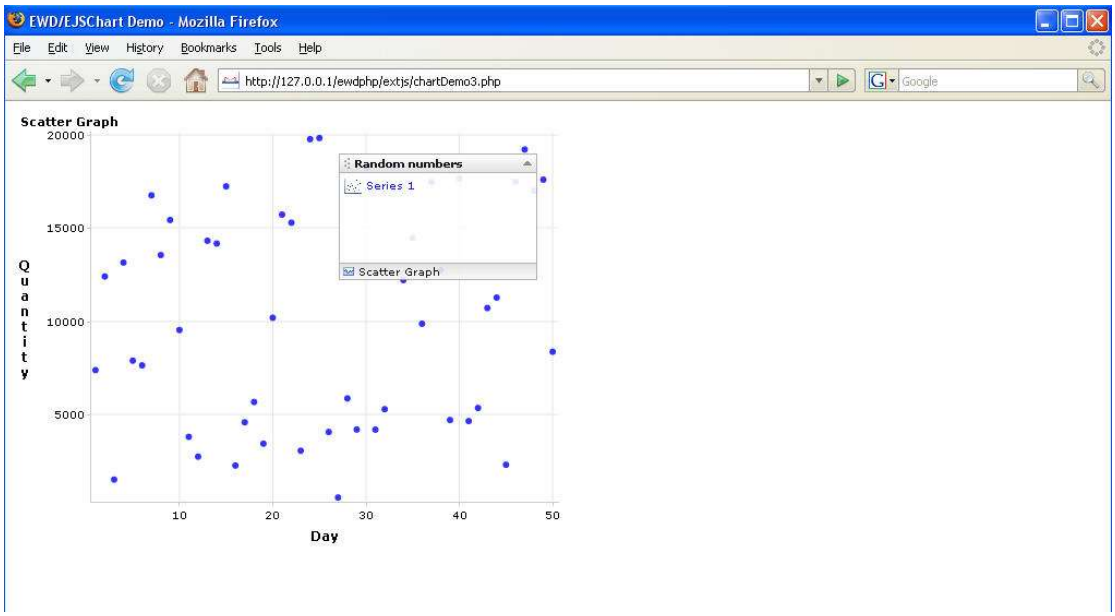
Note that you can combine line and area series together as required.

Scatter Series

The scatter series is also very similar to the line and area series and uses exactly the same datastore. So we can just adapt the *chartDemo2.ewd* page to create *chartDemo3.ewd*:

```
<ewd:config isFirstPage="true" prepagescript="##class(ewd.test).getDemoChart1Data"/>
<ejsc:config path="/EJSChart" />

<html>
  <head>
    <title>EWD/EJSChart Demo</title>
  </head>
  <body>

<ejsc:chart id="chart1" width="500px" height="400px" show_legend="true"
allow_interactivity="false" legendTitle="Random numbers" show_mouse_position="false"
title="Scatter Graph" x_axis_caption="Day" y_axis_caption="Quantity">
  <ejsc:scatterSeries id="chart2Series1" datastore="myArr1" color="rgb(0,0,250)"
pointStyle="circle" pointSize="3"  />
</ejsc:chart>

  </body>
</html>
```

We can use the same prePageScript (although we'll only be using the first datastore it creates). So save the new page, compile it and load the compiled version into your browser. You should now see something like this:



Note that by adding the trend attribute to a scatter series, you can add various trend lines, including linear, exponential and logarithmic. See the EJSChart API documentation.

Bar Series

The bar series allows you to plot horizontally or vertically aligned histograms. By combining multiple bar series into the same chart, you can also show multi-column histograms.

EWD makes it all very simple!

Create a new EWD page named *chartDemo4.ewd* in the *ejscDemo* directory that contains the following:

```
<ewd:config isFirstPage="true" prepagescript="##class(ewd.test).getDemoChart4Data"/>
<ejsc:config path="/EJSChart" />

<html>
  <head>
    <title>EWD/EJSChart Demo</title>
  </head>
  <body>

<ejsc:chart id="chart6" width="600px" height="400px" show_legend="false" title="Bar Chart"
x_axis_caption=" " y_axis_caption="Quantity">
  <ejsc:barSeries id="chart4Series1" datastore="myArr3" useColorArray="true" linewidth="5"
orientation="vertical"/>
</ejsc:chart>

  </body>
</html>
```

Compile this page as usual.

Now create the class method for the prePageScript as follows:

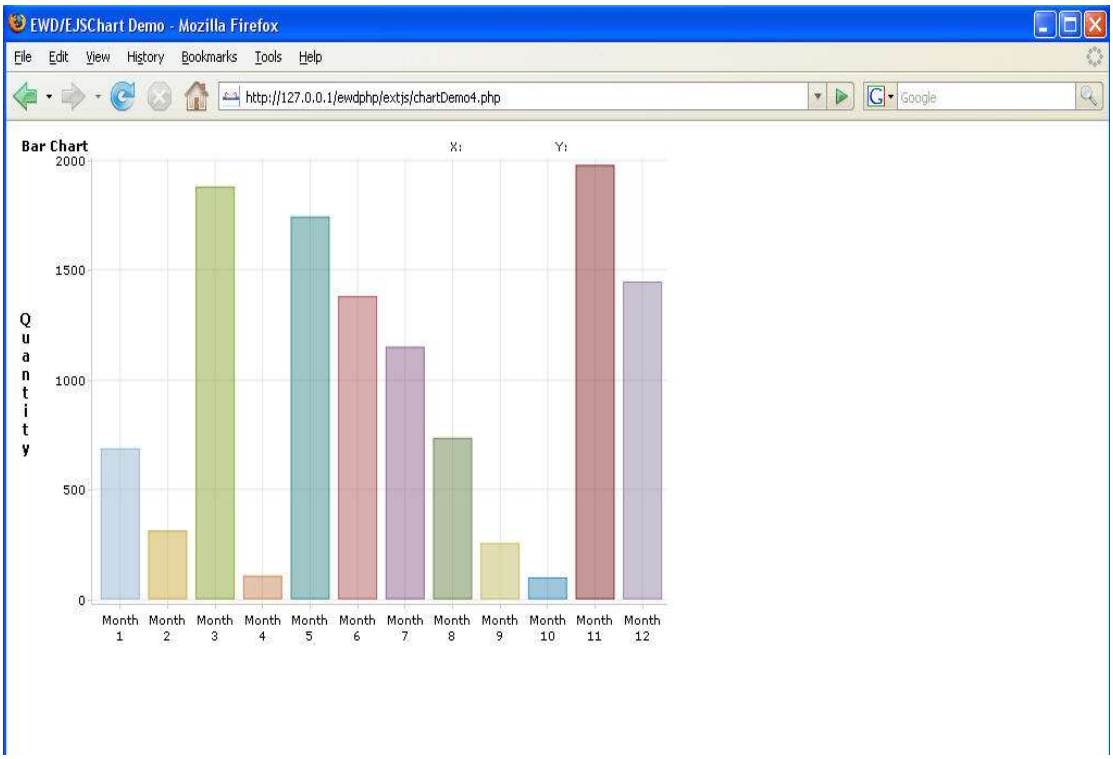
```
ClassMethod getDemoChart4Data(sessid As %String) As %String
{
f i=1:1:12 s data(i)="Month "_i_$c(1)_$r(2000)
d createDataStore^%zewdEJSC(.data,"myArr3",sessid)
QUIT ""
}
```

The structure of the bar series datastore is a little different. The value for each bar consists of two pieces (with ASCII 1 as the delimiter):

- the text to display under each bar
- the height of the bar

Save and compile this method.

Now display the compiled *chartDemo4* page in your browser and you should see something like the following:



In this example the bars are coloured differently because we specified *useColorArray="true"* in the *<ejsc:barSeries>* tag. We could apply a consistent colour to all the bars instead by replacing this with *color="rgb(0,0,250)"*. This is recommended if you are plotting two or more bar series within the same chart.

Pie Series

The pie series allows you to plot pie charts. Here's how to create them.

Create a new EWD page named *chartDemo5.ewd* in the *ejscDemo* directory that contains the following:

```
<ewd:config isFirstPage="true" prepagescript="##class(ewd.test).getDemoChart5Data"/>
<ejsc:config path="/EJSChart" />

<html>
  <head>
    <title>EWD/EJSChart Demo</title>
  </head>
  <body>

<ejsc:chart id="chart5" width="400px" height="300px" show_legend="false" title="Pie Chart">
  <ejsc:pieSeries id="chart5Series1" datastore="myArr" />
</ejsc:chart>

  </body>
</html>
```

Compile this page as usual.

Now create the class method for the prePageScript as follows:

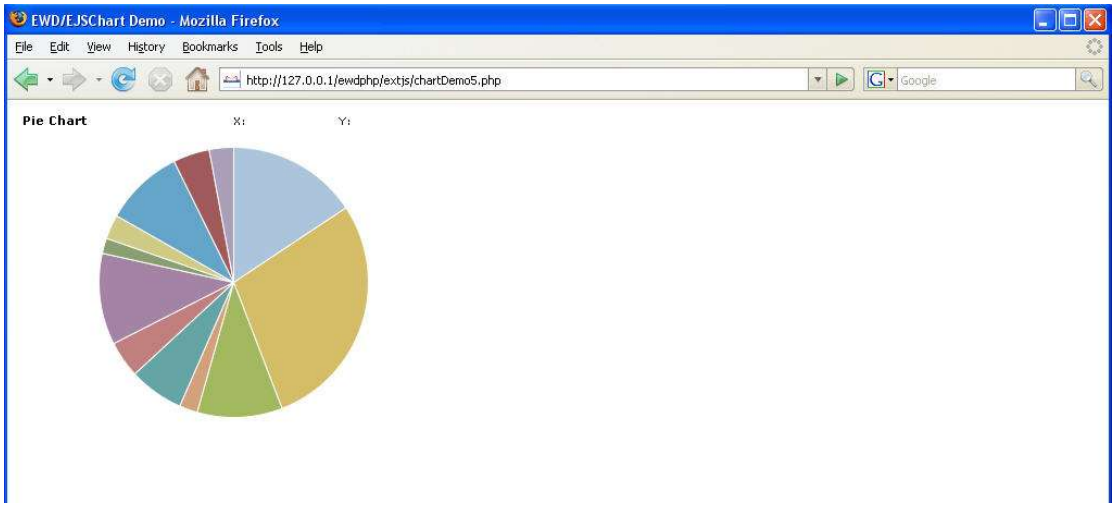
```
ClassMethod getDemoChart5Data(sessid As %String) As %String
{
f i=1:1:12 s data("Month" _i)=$r(100)
d createPieDataStore^%zewdEJSC(.data,"myArr",sessid)
QUIT ""
}
```

The structure of the pie series datastore is different again. The array element holding the value of each segment should be subscripted by the segment title, eg:

data("Month 8") = 29

Note that you must then put your array into a special EWD/EJSChart method named *createPieDataStore()*. This ensures that the correct JSON structure is applied to this data series.

Save and compile this class method and you should be ready to display the page chartDemo5 in your browser. You should see something like the following:



Dynamically Re-Drawing a Series

You can dynamically modify the data in a datastore and get EJSChart to re-draw the series without refreshing the page (or fragment). The trick is to use the EWD EventBroker.

Here's an example.

Let's go back to our first line series example:

```
<ewd:config isFirstPage="true" prepagescript="##class(ewd.test).getDemoChart1Data"/>
<ejsc:config path="/EJSChart" />

<html>
 <head>
  <title>EWD/EJSChart Demo</title>
 </head>
 <body>

<ejsc:chart id="chart1" width="500px" height="400px" show_legend="true"
allow_interactivity="false" legendTitle="Random numbers" show_mouse_position="false"
title="Line Graph" x_axis_caption="Day" y_axis_caption="Quantity">
 <ejsc:lineSeries id="chart1Series1" datastore="myArr1" drawPoints="true" title="My test series
1" color="rgb(0,0,250)" />
</ejsc:chart>

 </body>
</html>
```

First you need to determine the event that will trigger the re-draw. Let's start with a simple onClick event triggered by a button, eg add the following just before the </body> tag at the end of the page:

```
<br /><br />
<input type="button" value="Redraw" onClick="redraw()" />
```

Now add the Javascript function:

```
  <script language="javascript">
   function redraw() {
     ewd:##class(ewd.test).getDemoChart1Data() ;
     EWD.utils.getObjectFromSession('myArr1',true,false);
     try {
       chart1Series1.getDataHandler().setArray(myArr1);
       chart1Series1.reload();
     }
     catch(err) {
       return ;
     }
   }
  </script>
```

This function firsts re-invokes the back-end method used in the prePageScript, which produces a new set of random numbers in the datastore named *myArr1*. It then fetches this new instance of the JSON datastore into the browser using the EWD function:

```
EWD.utils.getObjectFromSession('myArr1',true,false);
```

Now we can get EJSCharts to redraw the series by using the commands:

```
chart1Series1.getDataHandler().setArray(myArr1);
chart1Series1.reload();
```

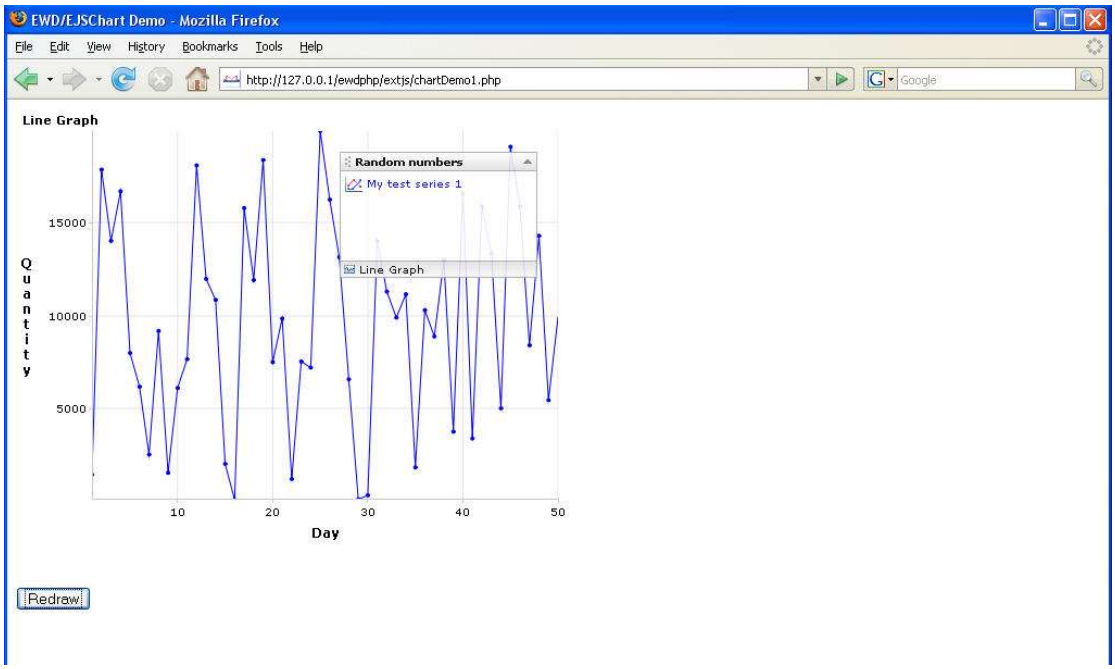
Note: we apply the *getDataHander()* and *reload()* methods to the *chart1Series1* object which is created because we specified an *id="chart1Series1"* in the <*ejsc:lineSeries*> tag, ie:

```
<ejsc:lineSeries id="chart1Series1" datastore="myArr1" drawPoints="true" title="My test series
1" color="rgb(0,0,250)" />
```

So, to recap, your amended page should now look like the following:

```
<ewd:config isFirstPage="true" prepagescript="##class(ewd.test).getDemoChart1Data"/>
<ejsc:config path="/EJSChart" />

<html>
 <head>
  <title>EWD/EJSChart Demo</title>
  <script language="javascript">
   function redraw() {
     ewd:##class(ewd.test).getDemoChart1Data() ;
     EWD.utils.getObjectFromSession('myArr1',true,false);
     try {
       chart1Series1.getDataHandler().setArray(myArr1);
       chart1Series1.reload();
     }
     catch(err) {
       return ;
     }
   }
  </script>
 </head>
 <body>

<ejsc:chart id="chart1" width="500px" height="400px" show_legend="true"
allow_interactivity="false" legendTitle="Random numbers" show_mouse_position="false"
title="Line Graph" x_axis_caption="Day" y_axis_caption="Quantity">
 <ejsc:lineSeries id="chart1Series1" datastore="myArr1" drawPoints="true" title="My test series
1" color="rgb(0,0,250)" />
</ejsc:chart>

<br />
<br />
<input type="button" value="Redraw" onClick="redraw()" />


 </body>
</html>
```

Recompile the page and load it into your browser. You should now see something like the following:



Each time you click the *Redraw* button, the line series should now change!

Strictly speaking we don't need the try/catch wrapper around these commands, but they can become necessary if we use a timer event instead of an onClick event. For example, change the page as follows:

```
<ewd:config isFirstPage="true" prepagescript="##class(ewd.test).getDemoChart1Data"/>
<ejsc:config path="/EJSChart" />

<html>
 <head>
  <title>EWD/EJSChart Demo</title>
  <script language="javascript">
   function redraw() {
     ewd:##class(ewd.test).getDemoChart1Data() ;
     EWD.utils.getObjectFromSession('myArr1',true,false);
     try {
       chart1Series1.getDataHandler().setArray(myArr1);
       chart1Series1.reload();
     }
     catch(err) {
       return ;
     }
        setTimeout("redraw()",freq);
   }
   freq=5000;
   setTimeout("redraw()",freq);
  </script>
 </head>
```

```
  <body>

<ejsc:chart id="chart1" width="500px" height="400px" show_legend="true"
allow_interactivity="false" legendTitle="Random numbers" show_mouse_position="false"
title="Line Graph" x_axis_caption="Day" y_axis_caption="Quantity">
 <ejsc:lineSeries id="chart1Series1" datastore="myArr1" drawPoints="true" title="My test series
1" color="rgb(0,0,250)" />
</ejsc:chart>

  </body>
</html>
```

Now the line series should automatically re-draw every 5 seconds. If this had been implemented inside an EWD fragment that was replaced by another, a timed event would be left running and would eventually attempt to invoke the getDataHandler() and reload() methods for a now non-existent object. By using the try/catch wrapper, this error will be safely trapped.

The Analog Guage Series

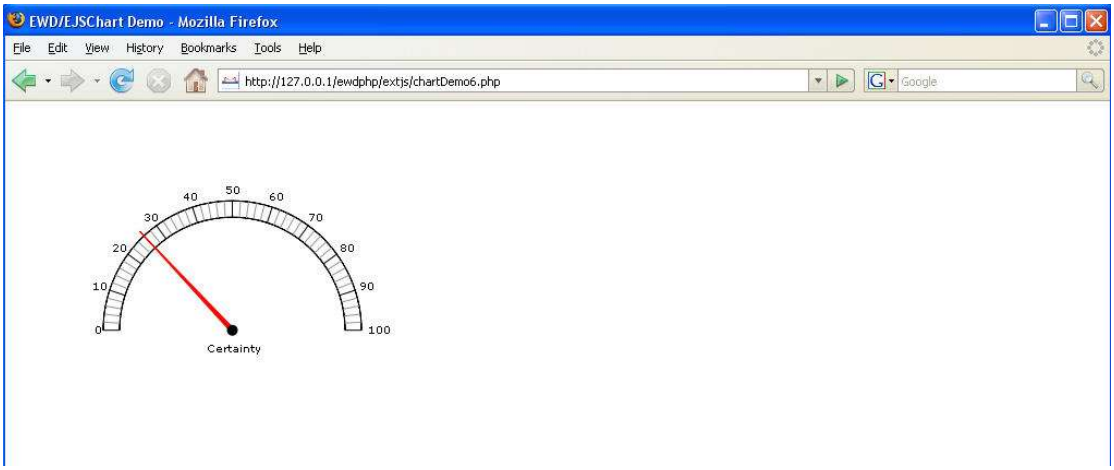
The last of the EJSChart widgets is the Analog Guage Series. This is not currently a core part of the EJSChart widget set and to make use of it you must specifically ask for it to be loaded into your main or container page. You do this by adding an additional attribute (*includeGuage*) to the <ejsc:config> tag:

```
<ejsc:config path="/EJSChart" includeGuage="true" />
```

The analog guage displays a single value, so its interfacing is via a simple scalar literal or EWD session value. For example, create a new EWD page named *chartDemo6.ewd* in your *ejscDemo* directory containing the following:

```
<ewd:config isFirstPage="true" />
<ejsc:config path="/EJSChart" includeGuage="true" />

<html>
  <head>
    <title>EWD/EJSChart Demo</title>
  </head>
  <body>

<ejsc:chart id="chart6" width="400px" height="300px" id="myChart" chartName="theChart"
show_legend="false" show_titlebar="false" show_x_axis="false" show_y_axis="false"
show_messages="false">
  <ejsc:analogGaugeSeries id="myGauge" dataStore="myStore" name="Certainty" value="26" />
</ejsc:chart>

  </body>
</html>
```

In this example we're assigning a fixed value of 26 that will be displayed in the guage. Compile the page and load it into your browser and you should see something like this:



Normally you'll want to get a value from your database or from some measurement observed within your back-end Cache system. In this case we'd use a session value assigned in the page's pre-page script. So, add a prePageScript to the example page and replace the *value=”26”* with a reference to the session variable that we'll create in the PrePageScript:

```
<ewd:config isFirstPage="true" prePageScript="##class(ewd.test).getDemoChart6Data"/>
<ejsc:config path="/EJSChart" includeGuage="true" />

<html>
  <head>
    <title>EWD/EJSChart Demo</title>
  </head>
  <body>

<ejsc:chart id="chart6" width="400px" height="300px" id="myChart" chartName="theChart"
show_legend="false" show_titlebar="false" show_x_axis="false" show_y_axis="false"
show_messages="false">
  <ejsc:analogGaugeSeries id="myGauge" dataStore="myStore" name="Certainty"
value="#myValue" />
</ejsc:chart>

  </body>
</html>
```

and create the class method:

```
ClassMethod getDemoChart6Data(sessid As %String) As %String
{
d setSessionValue^%zewdAPI("myValue",$r(100),sessid)
QUIT ""
}
```

This simply assigns a random number between 1 and 99 to the session variable named *myValue*.

Now when you recompile the page and reload it into the browser, you should see a random value appear in the analog guage. Every time you refresh the page, you should get a different value.

You can apply the timed redraw technique to the analog dial. The process is nearly the same as described in the previous section. Simply add the following Javascript function to your example page:

```
<script language="javascript">
  function update() {
    ewd:##class(ewd.test).getNewGuageValue() ;
    var myStore = [[guageValue,'Certainty']];
    try {
      myGauge.getDataHandler().setArray(myStore);
      myGauge.reload() ;
    }
    catch(err) {
      return ;
    }
    setTimeout("update()",3000) ;
  };
  setTimeout("update()",3000) ;
</script>
```

Then create a Caché class method as follows:

```
ClassMethod getNewGuageValue(sessid As %String) As %String
{
QUIT "guageValue="_$r(100)_" ;"
}
```

Recompile your example page and reload it into the browser. Now you should see the guage change to a different random value every 3 seconds!

Note the way we're manually re-setting the simple datastore used by the guage with the value returned via the Event Broker call to the Caché class method :

```
var myStore = [[guageValue,'Certainty']];
```

Using EJSChart Widgets inside EWD Ajax Page Fragments

EWD allows you to use any of the EJSChart widgets inside page fragments. The only thing you must remember to do is to add the <ejsc:config> tag to the top of the main container page, so that the EJSChart Javascript files are already loaded into the browser.

*Note: your EWD Ajax page fragments should **not** include the <ejsc:config> tag.*

Conclusions

That's all there is to using the amazingly powerful EJSChart widgets in your EWD pages and Ajax applications. You can freely mix and match them with ExtJS widgets, the combination giving you a genuinely rich set of user interface components.

With ExtJS and Emprise Javascript Charts, your EWD applications will never look the same!