Voyaging Deeper Into the BBjCharts API

A Closer Look Into *A Tour of the BBj Charts*

By Shaun Haney

While Florence found "ready-to-use" BBj® charts quite useful, she wanted to customize her existing charts and create some different types. The BBjCharts API (Application Programming Interface) provides these capabilities and more. This e-supplement explores:

- the base class of the BBjCharts API class hierarchy,
- how to customize the "ready-to-use" charts, and
- how to use BBjGenericChart to take advantage of the dozens of additional charts provided in the JFreeChart API.

BBjChart: The Base Class of the BBjCharts API

The BBjCharts API is an object-oriented API, complete with a top-level abstract class. Though a BBj developer cannot use the abstract class directly, there is a list of methods common to all the chart classes inherited from BBjChart. A BBjChart encapsulates these two objects:

- Dataset - the set of values for the chart to represent graphically
- ClientChart - the graphical representation of the chart

BBjBarChart, BBjLineChart, and BBjPieChart automatically manage these objects, making them fast and convenient to use. BBjGenericChart complements these basic charts, providing flexibility by allowing the developer to manage these objects directly. The methods in BBjChart are:

RETURN VALUE	METHOD
void	**writePNGToServer**(String p_filename)
void	**writePNGToServer(String** p_filename, **boolean** p_writeAlpha,**int** p_compression)
org.jfree.chart.JFreeChart@	**getClientChart()**
void	**clearData()**

The "@" symbol in the return value for the new BBj 7.0 feature getClientChart indicates that an object is a ClientObject. ClientObjects are server-side RMI (Remote Method Invocation) handles to client-side objects. For example, the return value of getClientChart is an RMI handle to an instance of org.jfree.chart.JFreeChart that resides on the client. This return value can call methods of the client-side JFreeChart when needing more control over the chart than what the BBjCharts API directly provides.

Because BBjChart is a BBjControl, the developer can make it focusable and tab traversable. The mouse events that BBjChart supports include entering and exiting the control, right-clicking, gaining and losing focus, and issuing a pop-up menu request on the control.

Customizing the "Ready-to-Use" Charts

BBjPieChart, BBjLineChart, and BBjBarChart all have the getClientChart method that allows the BBj developer to make direct calls to the underlying JFreeChart's methods. By calling these methods, the developer is able to take advantage of JFreeChart's robust interface. For example, Florence wants to add the title "Portion of Sales from each Revenue Source" and the subtitle "With a Focus on E-Commerce." Adding a title to the chart is as simple as calling BBjPieChart::setTitle(). But, how are subtitles added to the chart? BBjPieChart does not have a method for adding subtitles, but JFreeChart has a method called addSubtitles.

In order to call this method, first obtain the JFreeChart from the BBjPieChart as follows:

```
declare org.jfree.chart.JFreeChart@ underlyingChart!
underlyingChart!=pieChart!.getClientChart()
```

Next, prepare to call setSubtitles on underlyingChart!. The call to setSubtitles requires a list of titles, so create that list, thenCreate a java.util.ArrayList since this Java list is easy to work with. It is important to note that the ArrayList needs to be a ClientObject because it is an argument for another ClientObject's method, shown below:

```
declare java.util.ArrayList@ subTitleList!
subTitleList!=new java.util.ArrayList@()
```

continued...

Shaun Haney
Quality Assurance Engineer

From the online JFreeChart documentation, we quickly discover that a title is not simply a string, but rather an org.jfree.chart.title.Title**.** This class is just an abstract class derived by many different title classes that the developer can lay out in very specific ways, display images, etc. For a simple title that displays some text, use the class org.jfree.chart.title.TextTitle, described in the JfreeChart documentation under the heading "Direct Known Subclasses." Since the TextTitle object is going to be a parameter to the add method of the ArrayList, it also needs to be a ClientObject. To create a subtitle, add it to the ArrayList, and set it as a subtitle in the JfreeChart, enter the following code:

```
declare org.jfree.chart.title.TextTitle@ subTitle!
subTitle!=new org.jfree.chart.title.TextTitle@("With a Focus on E-Commerce")
subTitleList!.add(subTitle!)
underlyingChart!.setSubtitles(subTitleList!)
```

With the code added to the program, the simple BBjPieChart example discussed in *A Tour of the BBj Charts* now has a title and subtitle. Executing the downloadable sample subtitle.src results in the chart shown in Figure 1 that displays the desired subtitle.

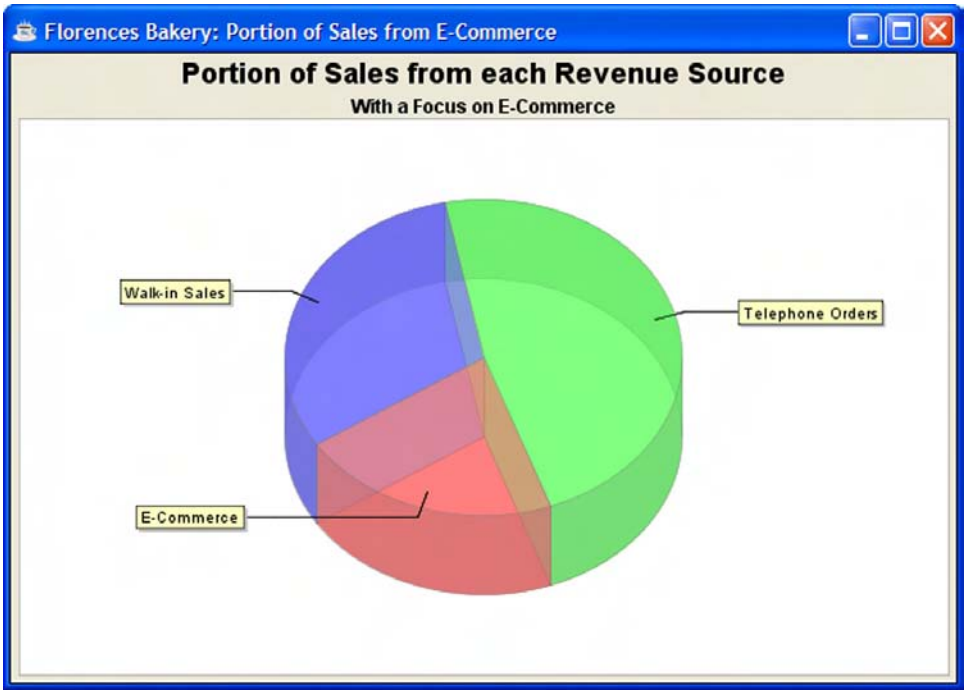


Figure 1. The sample subtitle.src customizes a BBjPieChart by adding a subtitle

The BBjGenericChart: A "Portal" into the JFreeChart API

It is quite convenient that BBj now has a bar, line, and pie chart immediately available, but what about using a different kind of chart?

Developers can use any of the dozens of different charts in the JFreeChart API for Java. This library is an open source project maintained by www.jfree.org in which individuals and interested companies continually add more chart types. BBjGenericChart harnesses the breadth and power of this library in BBj.

BBjGenericChart accepts any chart created via the org.jfree.chart.ChartFactory static factory methods. BBjGenericChart also allows access to the ClientChartPanel@, giving control of the chart's rendering and the range of data that appears in the chart. The developer seldom needs to manipulate the ClientChartPanel because the JFreeChart API provides reasonable defaults for zooming in to the chart and adjusting the range of data. By default, the charts display the entire range of the data in the dataset; clicking and dragging bounding boxes within the chart displays a specific range. A simple click-and-drag to the left returns the chart to its original scale.

To use the BBjGenericChart, follow these steps:

1. Create the BBjGenericChart using the BBjWindow::addGenericChart() method
2. Create the dataset for the chart and populate the dataset with the data
3. Instantiate the JFreeChart with the dataset
4. Set the BBjGenericChart's ClientChart to the newly created JFreeChart

continued...

To understand these steps more clearly, take a closer look at them in action.

Create the BBjGenericChart

Mardi Gras 2007 was in March so Florence is interested in seeing the March's e-commerce sales in a ring chart - a pie chart with a hollowed-out middle that ironically bears resemblance to the season's traditional king cake. Following the steps listed above, create the BBjGenericChart using the BBjWindow::addGenericChart() method. Just like other BBjControls, it has a control ID, a top-left position, and a width and height shown below:

```
chartControl!=win!.addGenericChart(101,0,0,640,480)
```

Create the Dataset with a Brief Detour into ClientObjects

Next, create the dataset for the ring chart. The ring chart's dataset is rather simple compared to some of the datasets in the JFreeChart library. The dataset consists of "slices" where each slice has a unique name and value. Just like the pie chart, the ring chart displays each ring segment based on the proportion its value is of the total as shown in **Figure 2**.

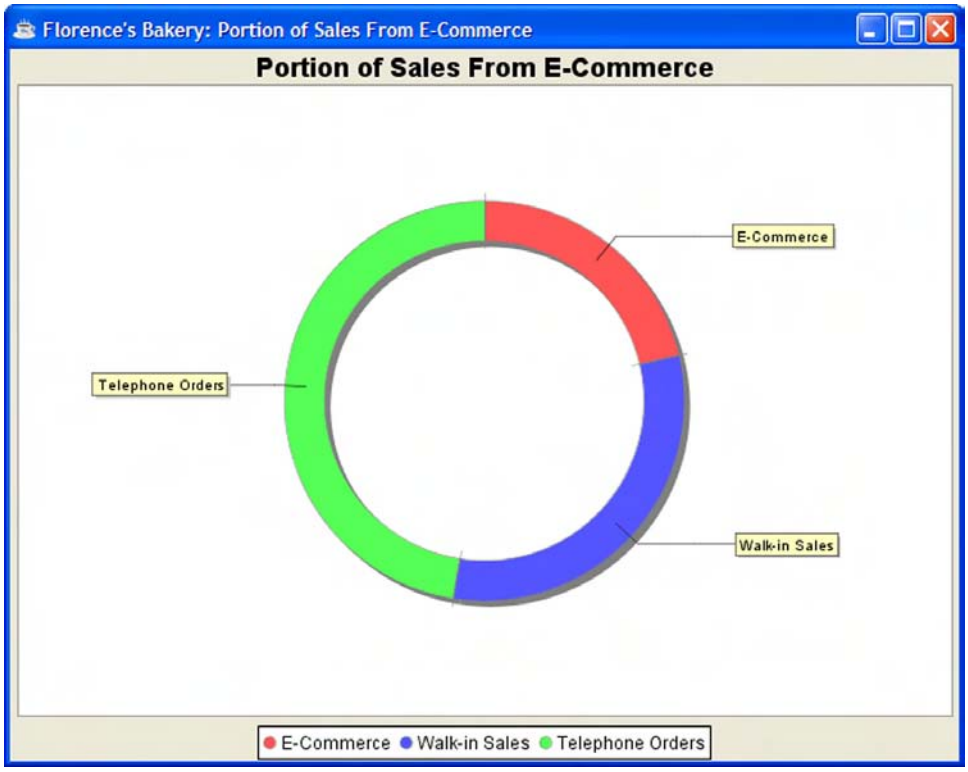


Figure 2. Sales data displayed in a JFreeChart ring chart

It is important to note, however, that the newly created chart is a client-side object and any objects that use it must be client-side objects as well. BBj normally renders any GUI object that is not a BBjControl on the server, not the client. If BBj Services is running on the same machine as the BBj program, the discrepancy may not be obvious. However, as soon as BBj Services runs on a different machine than the BBj program, the GUI objects will not appear because BBj renders them using the server's JVM.

To instruct BBj to use the client's JVM for rendering the graphical objects, use client-side objects. Create these objects by attaching the @ symbol to its type name to indicate "the client-side version of this type."

The following sample demonstrates how to get the client's local time using Client Objects. Even if BBjServices is running several timezones away, this program will show the time on the current system. Without using Client Objects, this same program would show the time on the machine running BBjServices:

```
Use java.util.Date
declare Date@ currentTime!
currentTime!=new Date@()
print currentTime!.getHours(),":",currentTime!.getMinutes(),":",
:currentTime!.getSeconds()
```

continued...

The only time the @ symbol is utilized is when referring to a specific variable's type. The use statement syntax does not include an @ symbol with the typename, but declarations and instantiations require it for clientside objects. The @ symbol is not used as part of the variable's name, so calling methods from **currentTime!** works the same way as calling methods from other object variables.

In order to create the dataset for the ring chart, first reference the DefaultPieDataset with this statement:

```
use org.jfree.data.general.DefaultPieDataset
```

Next, declare the dataset in the following manner:

```
declare DefaultPieDataset@ dataset!
```

Then, instantiate the dataset and fill it with data for each of the three sources of revenue such as:

```
dataset!=new DefaultPieDataset@()
dataset!.setValue("E-Commerce",5031.98)
dataset!.setValue("Walk-in Sales",7350.68)
dataset!.setValue("Telephone Orders",11205.51)
```

Creating the JFreeChart
To create the JFreeChart, which is also a ClientObject, use a static factory method from org.jfree.chart.ChartFactory with the org.jfree.chart.JFreeChart and org.jfree.chart.ChartFactory classes. This saves time and eliminates possible errors by referring to the JfreeChart classes directly instead of by their fully qualified names. Skipping this step would require the developer to address ChartFactory as org.jfree.chart.ChartFactory throughout the code. For example, begin with the following use statements:

```
use org.jfree.chart.ChartFactoryuse org.jfree.chart.JFreeChart
```

Next, declare the JFreeChart variable as:

```
declare JFreeChart@ chart!
```

Finally, instantiate chart!. Creating a client-side JFreeChart object requires a call to createRingChart as a static method from the client-side version of ChartFactory shown below:

```
chart!=ChartFactory@.createRingChart("Portion of Sales From E-Commerce",dataset!,1,1,1)
```

Set BBjGenericChart's ClientChart
The next step is to set BBjGenericChart's ClientChart to the JFreeChart just created. Recall that BBjGenericChart is a BBjControl and not a client object. However, it does manage a JFreeChart, which is a client object. To set up the BBjGenericChart to manage the JfreeChart, call the BBjGenericChart::setClientChart() method this way:

```
chartControl!.setClientChart(chart!)
```

Following these steps displays the ring chart inside the BBjGenericChart control.

BBjGenericChart Example
Just like the BBjPieChart example, the code to create the ring chart displays the portion of sales each revenue source generated: walk-in sales, telephone orders, and e-commerce site transactions. The downloadable sample ringchart.src combines all the steps needed to create a ring chart using the BBjGenericChart control.

Another BBjGenericChart Example: The Candlestick Chart

Florence realized that another use for charts in her business was to display fluctuations in the bakery's bank accounts. Florence hoped to find that the closing balance for any given month was higher than the opening balance and that the downward fluctuation of the bank balance would be at a minimum to keep the bakery "in the black." As a result, Florence wanted to chart this in spite of her accountant's raised brow. To track such activity, Florence could use a candlestick chart. Though most often used to track stock prices, the candlestick chart could be a good fit for tracking the bakery's bank balances.

The Dataset
In the JFreeChart documentation, look at http://www.jfree.org/jfreechart/api/javadoc/org/jfree/chart/ChartFactory.html. Locate the createCandleStickChart method. In the method's signature, notice that one of the parameters the method takes

continued...

is an OHLCDataset. OHLC stands for "Open High Low Close." Click on the word "OHLCDataset." It turns out that OHLCDataset is just an interface. We will create one of the implementers of the OHLCDataset called the OHLCSeriesCollection. This dataset allows us to represent open, high, low, and close values for given intervals of time. In candlestick.src, we are tracking bank balances by months, which will be our interval. This dataset has a default constructor with no parameters so creation is quite simple:

```
use org.jfree.data.time.ohlc.OHLCSeriesCollection
declare OHLCSeriesCollection@ dataset!
dataset!=new OHLCSeriesCollection@()
```

The next step is to create a series for our dataset. A series is a set of data associated with a particular entity. In this case, the series we are creating is the opening balances, high balances, low balances, and closing balances month-by-month over 24 months for the bakery's bank balance. The series used by OHLCSeriesCollection is OHLCSeries. The only parameter requiredto instantiate the series is the name of the series. Here, the name of the series for the data it represents is the bakery's bank balance:

```
use org.jfree.data.time.ohlc.OHLCSeries
declare OHLCSeries@ series!
series!=new OHLCSeries@("Bank Balance")
```

Now that we have created the series, we are ready to populate it. OHLCSeries has an add method, which takes one unit of data. The unit of data is the time interval and the opening, high, low, and closing values for that time interval.
Before we call this method, we must create the time interval. The Month time interval has a constructor that takes two numbers: a year and a month (values between 1 and 12). For example, to create a month representing October of 2007, enter the following:

```
use org.jfree.data.time.Month
declare Month@ interval!
interval!=new Month@(10,2007)
```

Suppose that for the month of October, 2007, our opening balance is $2,500, our high balance is $6,300, our low balance is $3 and our closing balance is $1,200. Here's how we would add this data to our series:
series!.add(interval!,2500,6300,3,1200)

Once we have created the series, we are ready to add it to the dataset:

```
dataset!.addSeries(series!)
```

In this demo, we only use one series representing a single bank account. It is possible and would be simple matter to create several series, each representing a different bank to add to the dataset. Once the dataset is added to the candlestick chart, all the different series would be displayed.

Create the Chart
After creating the dataset, we have everything we need in order to instantiate the JFreeChart. The factory method for creating a candlestick chart requires the following parameters: title, name of the X axis, name of the Y axis, dataset, and whether to display the legend. We're going to title our chart "Bakery Budget Fluctuations;" name the X-axis "Months" since it is the axis used for time; name the Y-axis "Bank Balance in U.S. Dollars; " use the OHLCSeriesCollection we created; and choose to display a legend:

```
jfchart!=ChartFactory@.createCandlestickChart
:   "Bank Balance Fluctuations","Months","Bank Balance in U.S. Dollars",
:   dataset! ,1)
```

Change the Width of the "Candlesticks"
Now we have done everything necessary to create the candlestick chart and display it on the screen. However, by default, the candlesticks appear quite narrow. On some systems, it may even be difficult to tell what color each candlestick is; they all may appear red. It is possible to make the candlesticks wider so that they are more easily visible with the JFreeChart API's CandlestickRenderer. The following code does just that:

continued...

```
use org.jfree.chart.plot.XYPlot
use org.jfree.chart.renderer.xy.CandlestickRenderer
declare XYPlot@ plot!
declare CandlestickRenderer@ renderer!

plot!=cast(XYPlot@,jfchart!.getPlot())
renderer!=cast(CandlestickRenderer@,plot!.getRenderer())
renderer!.setCandleWidth(candleStickWidth)
```

The Candlestick chart in the JFreeChart API has an XYPlot, which in turn contains a CandlestickRenderer. The CandlestickRenderer has a method setCandleWidth, which allows the developer to set the candlestick width. Each chart in the JFreeChart API may have one or more plots, which may have one or more renderers. The factory methods, called to create charts using `ChartFactory`, set these up. The JFreeChart Developer Guide, mentioned at the end of this article, has more information on accessing the plots and renderers of the charts.

The Candlestick Chart Example

The resulting chart in **Figure 3** displays green blocks indicating that the closing balance is higher than the opening balance and red blocks indicating a lower closing balance than the opening balance. Lines rising upward from the block indicate that the high value for that date was greater than the opening and closing balances; lines falling from the block indicate that the low value was less than their opening and closing balances. Usually blocks in a candlestick chart have rising and falling lines, although Florence would like to see green blocks in her charts.

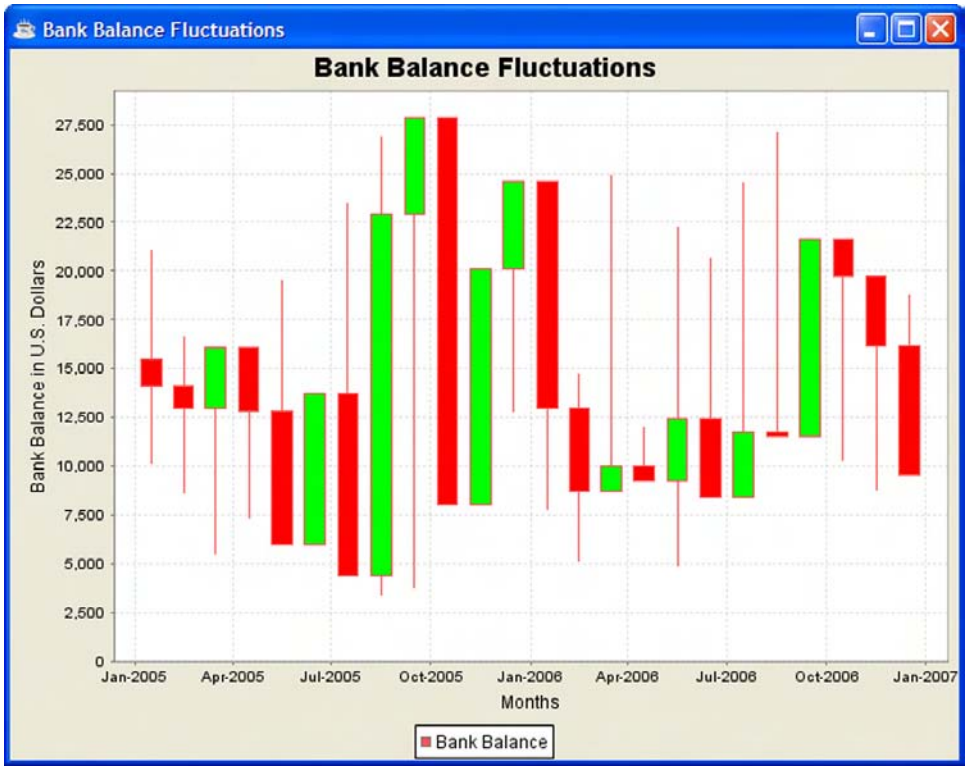
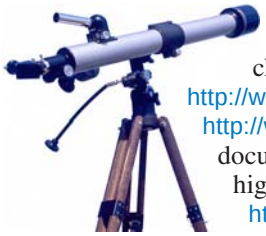


Figure 3. Candlestick chart showing bank balance fluctuations



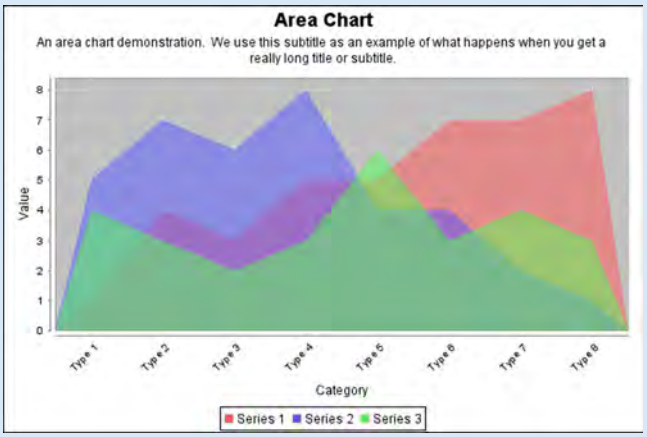
Other Resources

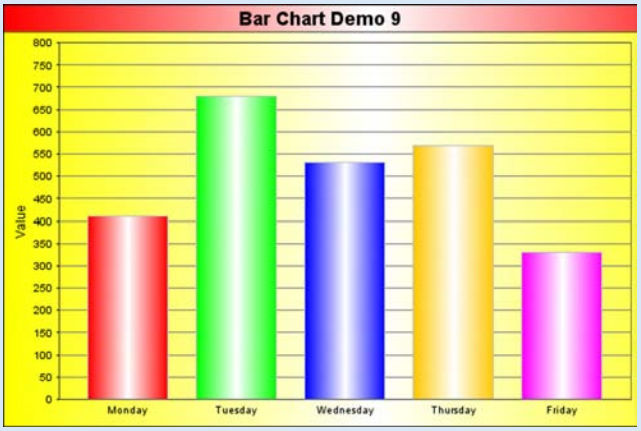
Currently, the JFreeChart API contains 28 different kinds of charts. Besides the standard bar, line, and pie charts, this API features scatterplots, Gantt charts, and charts that plot using polar coordinates. Go to http://www.jfree.org/jfreechart/samples.html to view all of the available charts via an interactive runnable demo. Visit http://www.jfree.org/jfreechart/index.html for a wealth of additional resources, to access free online API documentation, or purchase the JFreeChart Developer Guide for using the different datasets and charts. BBj is highly extensible, allowing developers to use Java objects in their code, so further explore the Java 6 API at http://java.sun.com/javase/6/docs/api/.

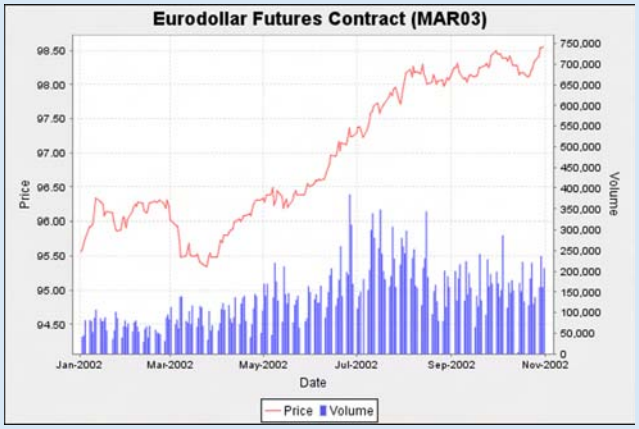
Summary

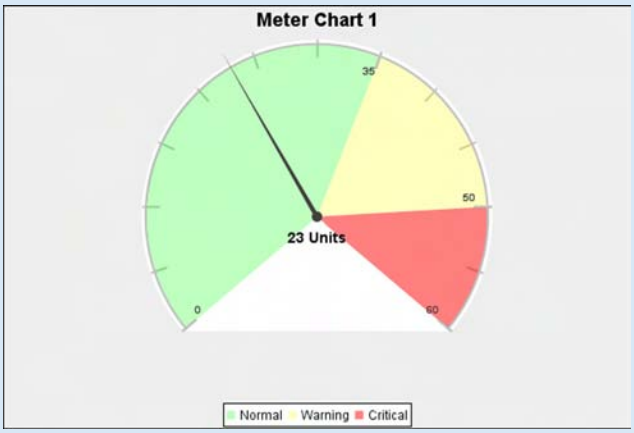
As Florence discovered, standard pie, line, and bar charts do not always "cut the cake." BBj allows developers to take advantage of a large variety of charts found in the JFreeChart API by enhancing BBj's ready-to-use charts via the ClientChart or by using the BBjGenericChart to create one of the dozens of other types of charts as shownin **Figure 4**. While creating these additional charts require a little more coding, they are highly customizable.

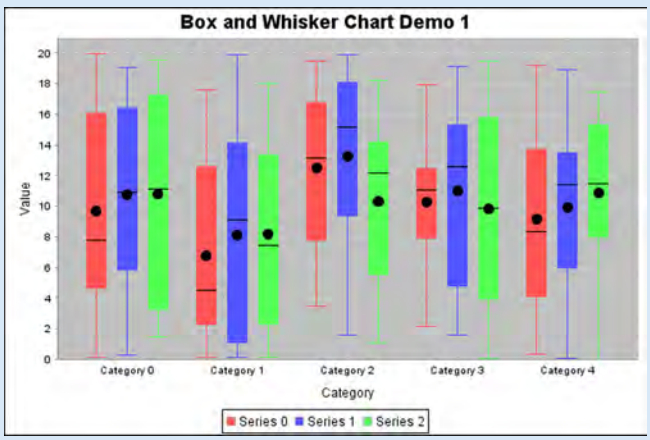
May you find the coding techniques detailed in this article useful in creating your own charts!

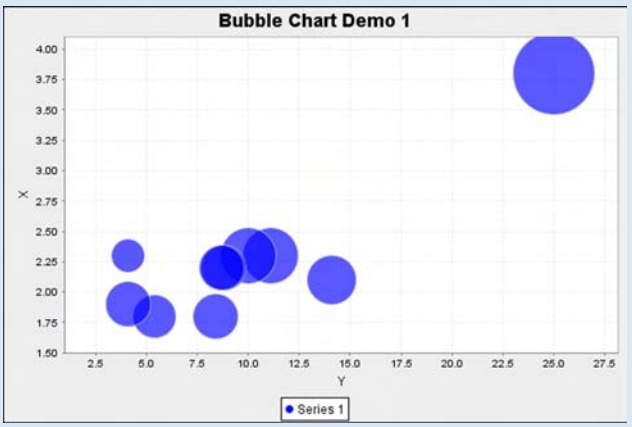


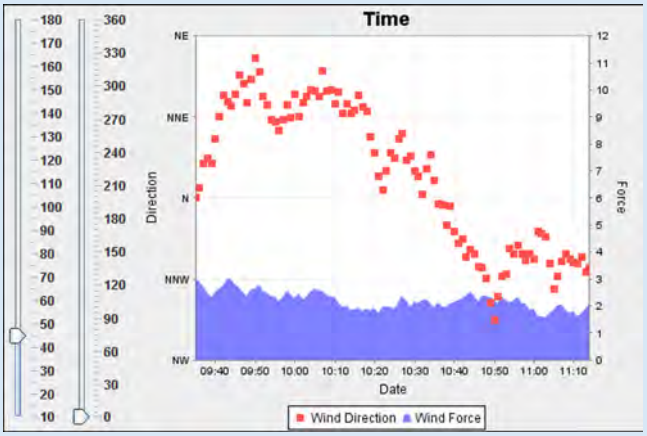












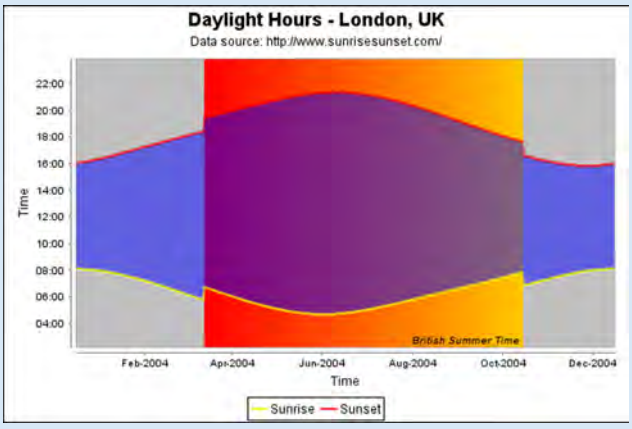


Figure 4. Example of other charts types