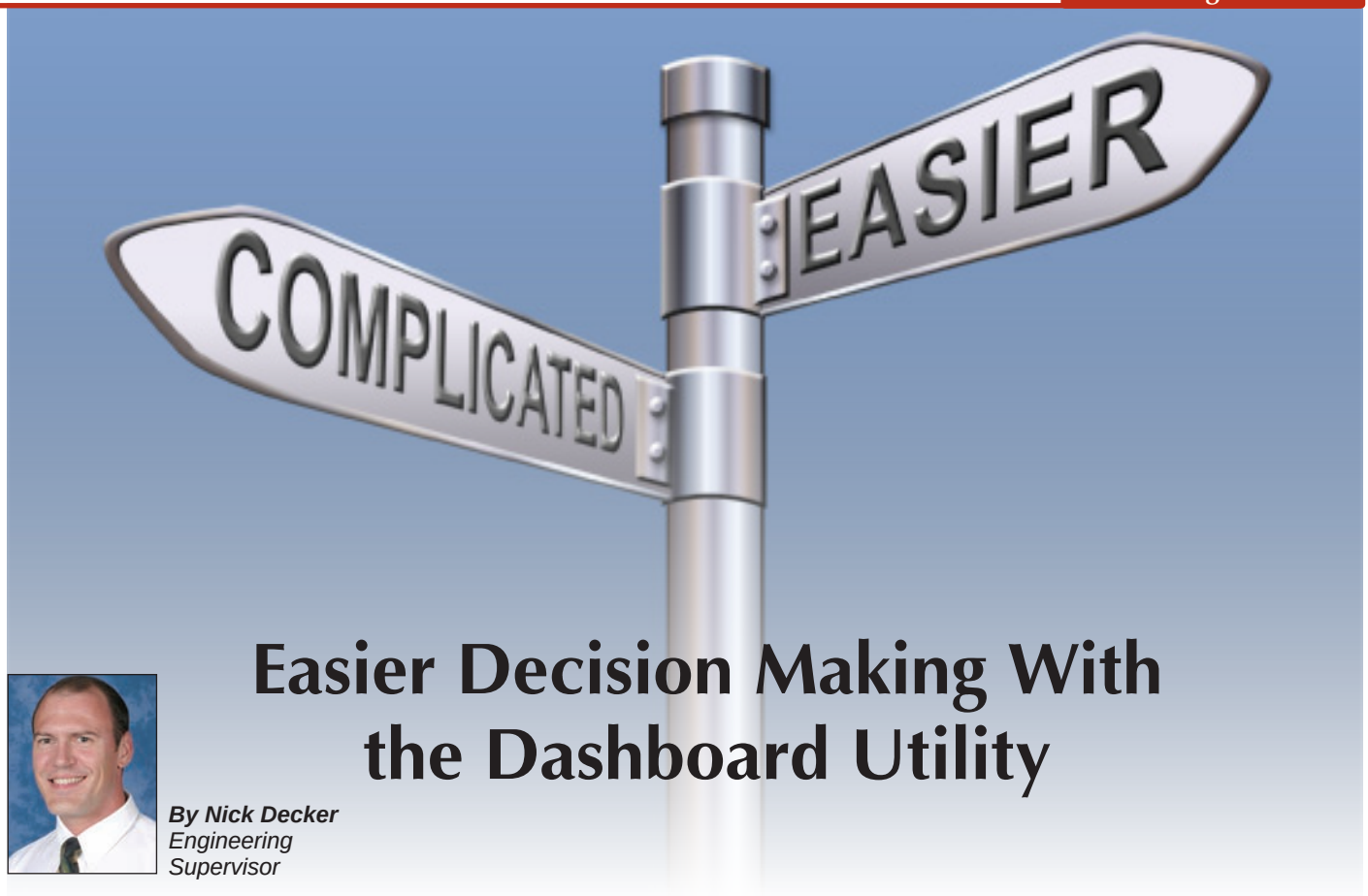


Easier Decision Making With the Dashboard Utility

By Nick Decker
Engineering Supervisor

The Dashboard Utility eases BBj developers' job in many ways, providing built-in functionality that would normally require significant coding time and development. The utility also simplifies application development by providing a higher-level API in addition to adding several useful niceties. This article takes an in-depth look at a few of the ways the Dashboard Utility reduces development time and facilitates building high quality dashboards in a modest amount of code. The Dashboard Utility delivers data in a format that will allow business owners and managers to turn decision making into child's play!

Layout is Easier

When you create a dashboard such as the one in **Figure 1** that we built in our Java Break *Adding the New Digital Dashboard to Your App*, you reap the benefits of a fully functioning layout system.

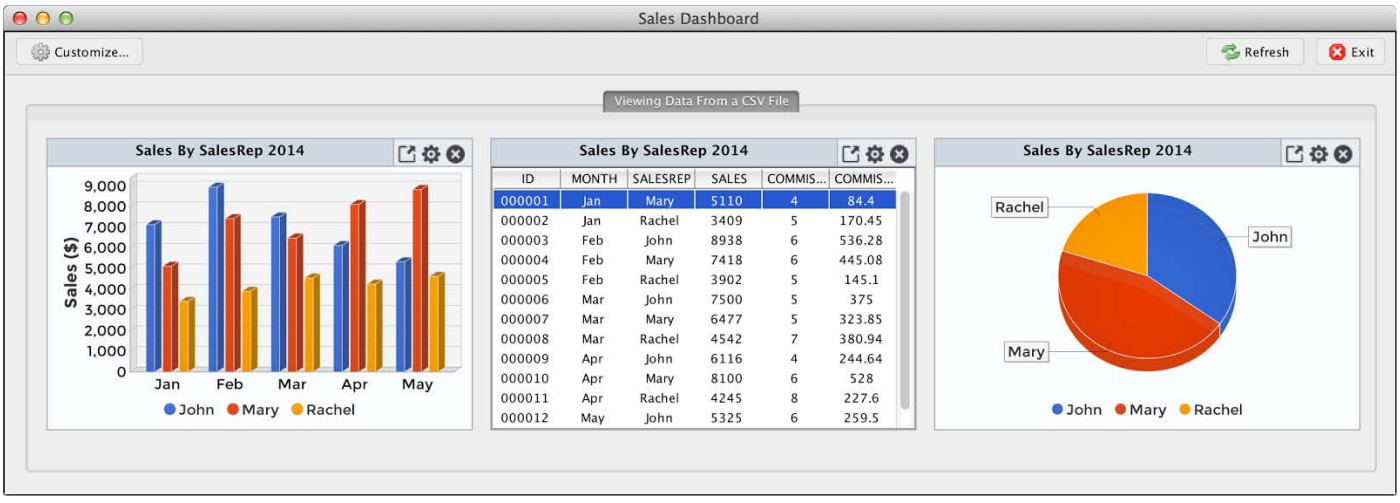


Figure 1. The dashboard program built during the Java Break

The DashboardControl is the top-level window that contains all of the dashboard elements, already programmed to handle events such as maximizing, minimizing, resizing, and positioning. Better yet, it saves these preferences so that the dashboard will be the same size and in the same position the next time you run it. Whenever you change its client area, such as maximizing or resizing the window, it internally calculates the new client area and the optimum size and placement for all of the widgets.

Dashboard widgets come with a default minimum and maximum width that the dashboard uses when calculating the number of rows and columns to show. The result is that the dashboard displays the widgets at a calculated size that makes the best use of the available client area. So depending on whether you resize the dashboard window to be tall and narrow, or short and wide, you may end up with two columns of three widgets each, or three columns of two widgets each. This means it is even possible to run the dashboard on a smartphone or other mobile device, as it resizes the window and widgets automatically to make the best use of the limited space. Because you can control the widgets' minimum and maximum size, along with the spacing between the rows and columns of widgets, you can influence the layout and customize it for a particular device, as shown in **Figure 2**.

Figure 2. The effects of changing the widget size for a smartphone running in portrait mode

The dashboard also responds to device orientation changes so you get a different layout in portrait and landscape mode. **Figure 3** shows the same dashboard program running in a smartphone in landscape orientation.

Figure 3. The dashboard running on a smartphone in landscape mode

Data is Easier

One of the more striking examples of how the API streamlines development deals with populating charts with data. For example, you would typically incorporate a BBjBarChart into an application with the following program flow:

1. Add the BarChart to the window via the addBarChart() method, providing several initial parameters.
2. Initialize data access from a file or database.
3. Retrieve the data in a loop.
4. Execute the setCategoryName() method to add a data category based on the data.
5. Execute the setSeriesName() method to add a data series based on the data.
6. Repeatedly execute the setBarValue() method to add data to the chart.

The Dashboard Utility takes over the onus of creating a window and handling its events, so instead of adding a BBjBarChart to a window via the addBarChart() method, your code will add a BarChartWidget to a DashboardCategory via the addBarChartDashboardWidget() method. These two methods are similar in theory and somewhat related in practice, as they both take parameters to indicate the chart's labels, orientation, dimensionality, etc. The difference is that you can populate the chart automatically by providing a BBjRecordSet or a database connection string and SQL query to the addBarChartDashboardWidget() method.

The utility takes on the task of populating the chart, saving you a lot of time, code, and effort. Of course, you can still create an empty BarChartWidget and populate it with data yourself, just as you did with the BBjBarChart. However, instead of using three different methods to add categories, series, and set values, you can accomplish the same thing with a single setDataSetValue() method call that adds the underlying categories and series for you automatically. The bar chart shown in **Figure 4** is a perfect example, as it was created with a connection to the Chile Company database and an SQL query that retrieved the top four customers ordered by their sales thus far this year.

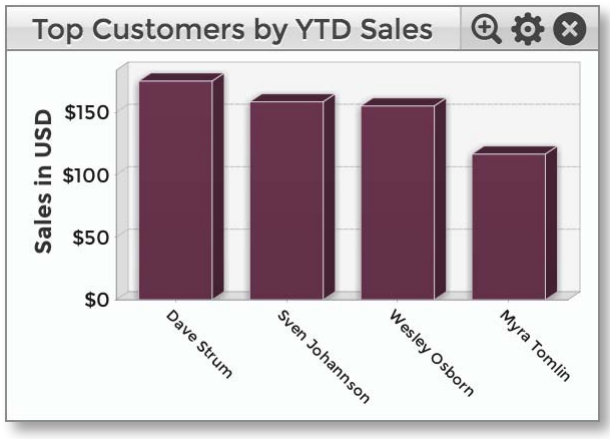


Figure 4. A dashboard widget that was filled automatically given an SQL connection and query

Formatting Grids is Easier

Other perks are sprinkled throughout the API, and our next example deals with formatting the appearance of a GridWidget. In many cases, you'll want to customize the width of a grid's columns to improve legibility by allocating more space to columns with more data. For example, when displaying customer addresses, the City column requires more space than the State column that only shows two-character abbreviations. To accomplish this with a typical BBjGrid control, you would make repeated calls to the setColumnWidth() method, providing the column number and the desired width in pixels.

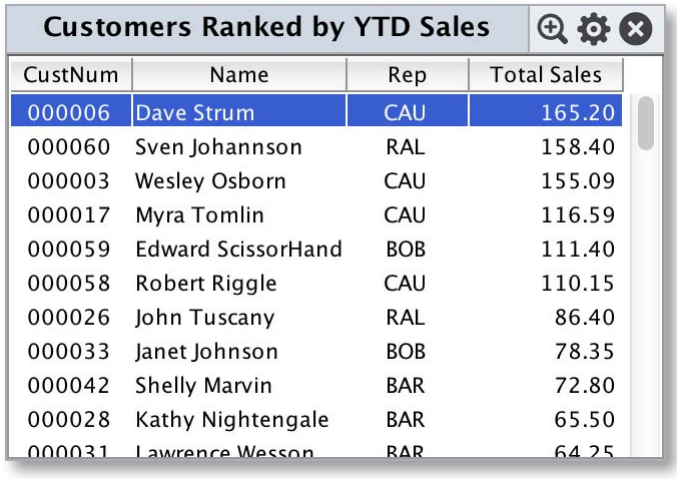
The dashboard's GridWidget is flexible, and often changes size depending on the size of the dashboard itself, a minimum or maximum widget size preference set by the developer, and whether or not you popped the widget out. Because of the various size possibilities, setting an absolute pixel width for a column isn't feasible. Instead you can call one of the setColumnWidthPercentages() methods to specify the width of all of the columns at once based on a percentage of the grid's width. The line of code below demonstrates setting the widths of all four of a GridWidget's columns at once using a comma-delimited string of percentage values, although another variation of the method exists that takes a BBjVector as the parameter. This line indicates that the first column should take up 20% of the grid's total width, the second column should take up 35%, and so on for a total of 100%.

```
gridWidget!.setColumnWidthPercentages("20,35,20,25")
```

The resultant grid's columns are sized perfectly as shown in **Figure 5**, even when the grid is resized or popped out and enlarged.

JFreeCharts are Easier

When working with a BBjChart, you could always get the underlying JFreeChart client object via the getClientChart() method. This allowed you to exercise literally thousands of methods against the underlying chart and its components such as its plot, renderer, legend, and all of their components. On the plus side, you had complete control over the resultant chart. On the minus side, the JFreeChart API is sufficiently deep and complex that in practice very few developers went through the effort to make any modifications at all. Additionally, since the getClientChart() method returns a client object, this means that it is not an option in BUI and therefore is even less likely to be used.



CustNum	Name	Rep	Total Sales
000006	Dave Strum	CAU	165.20
000060	Sven Johannson	RAL	158.40
000003	Wesley Osborn	CAU	155.09
000017	Myra Tomlin	CAU	116.59
000059	Edward ScissorHand	BOB	111.40
000058	Robert Riggle	CAU	110.15
000026	John Tuscany	RAL	86.40
000033	Janet Johnson	BOB	78.35
000042	Shelly Marvin	BAR	72.80
000028	Kathy Nightengale	BAR	65.50
000031	Lawrence Wesson	BAR	64.25

Figure 5. The result of setting the grid's column widths in percentages

In direct contrast, it is relatively simple to make dramatic and sweeping customizations to a dashboard chart, usually in just a couple of lines of code. For example, you can easily customize all of the chart's colors, either by selecting a pre-existing color theme or providing your own colors, in a single line of code. Likewise, you can also change the font

sizes or colors for all of the elements in a chart in a single line of code. In addition to being efficient, the high-level API relieves you from the effort required to drill down into the JFreeChart hierarchy to affect changes. This means you can change the colors on the chart widget itself, instead of getting the underlying chart, getting the plot from the chart, getting the renderer from the plot, and executing the setSeriesPaint() method on the renderer to set the colors.

The dashboard shown in **Figure 6** shows the result of chart customization. By taking advantage of methods on the widgets, we were able to set custom chart colors, modify the fonts, change the background color of the legend, add a drop shadow to the plots, and change the range axis to format the values as currency. All of this was possible in just a few lines of code on the widgets, whereas it would have taken a significant amount of low-level code to accomplish the same task on a JFreeChart object.

Figure 6. A dashboard with customized widgets

Summary

The BASIS Dashboard Utility curtails the amount of effort required to visualize your data effectively in a single widget or complete dashboard. Gone are the days where you would have to write database integration code in order to populate BBjCharts, as the Utility can automatically populate and refresh widgets for you. The Utility also handles other crucial tasks, such as sizing and positioning widgets, so your dashboard looks fantastic – even on mobile devices such as smartphones and tablets. If you have not done so yet, take the Dashboard Utility out for a spin by visiting our BUI Showcase page and running some of the Dashboard Demos on your favorite computing device! ■

- Watch the Java Break Adding the New Digital Dashboard to Your App on YouTube
- Refer to other articles in this issue
  - *Dash Boredom With the Dashboard Utility*
  - *The Magic of the Widget Wizard*
- Visit the online Help
  - Dashboard Utility Overview
  - Dashboard Javadocs