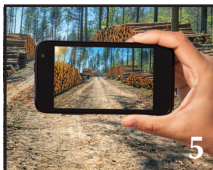


BASIS International Advantage

Volume 18 • Autumn 2014



22 more articles at
links.basis.com/14adv
in this 110+ page issue!

Create
Your
Masterpiece



Table of Contents

Partnership

- 5 A DAM-EDV Picture is Worth 1,000 Woods** by *Patrick Schnur*
Answer your curiosity about "how 'wood' one go about writing an app for a smartphone?" as you read about DAM-EDV's journey to deliver this innovative web-based mobile solution. [14dam-edv](#)



- 35 CarlIT Integrates a 'New Model' With BBj** by *Wimco Driesse*
Be inspired by this developer who tells, in his own words, how his company fully modernized their Microsoft technology app and their Visual PRO/5 app by converting and integrating them with BBj, paving the way for more future enhancements. [14audev](#)



Language/Interpreter

- 50 Bleeding Heart Computer Security** by *Dan Christman and Jerry Karasz*
Look out for security issues such as this one that affect apps using an OpenSSL library, and learn why BASIS remains vigilant about future risks. [14heartbleed](#)



- 68 The Anatomy of BBj** by *Teresa Dominguez*
Observe this dissection of BBj to better understand its components and how they connect to each other thereby contributing to its robust inner workings. [14bbj](#)



- 78 Add New Grid Selections to Your Toolbox** by *Aaron Wantuck*
Migrate to the new Enhanced Selection Model for better user control of the grid and a more efficient development process. [14grid](#)



- 101 Wash up With SOAP Web Services** by *Richard Stollar*
In this easy-to-follow tutorial, learn how to use more complex data structures with your web services and implement Basic authentication. [14webservices](#)



DBMS

- 19 Asynchronous Triggers Modify the Copy** by *Jeff Ash*
Preview how to execute specific BBj code whenever write and remove operations occur on a list of monitored directories and files, and monitored databases, without blocking the data access operation. [14triggers](#)



- 52 Replication Redux** by *Chris Hardekopf*
Read about the many improvements to the efficiency of your replication job, including maintaining large numbers of files being replicated from the source to the target. [14replication](#)



Development Tools

- 22 Have it Your Way With New BDT Preferences** by *Kevin Hagel*
Customize how you create and manage BBj projects to your own liking using a number of new advanced workspace preferences and project properties. [14bdtprefs](#)



- 38 BDT Tips for Less Pain and More Gain** by *Nick Decker*
Discover these features that BASIS' own engineers say you 'can't live without' in the BDT Eclipse plug-in CodeEditor, enhanced with BBj-specific capabilities. [14bdttips](#)



- 62 Test for Success With BBj Unit Test** by *Jerry Karasz and Sebastian Adams*
Catch those bugs and fix them easily before you deliver your app, using the new BBj Unit Test Eclipse plug-in to run unit tests against your BBj code. [14unittest](#)

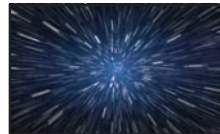


- 81 Building WindowBuilder** by *Jerry Karasz*
See into the future WB Eclipse plug-in with this Alpha version tool designed to help you lay out your graphical controls easily and quickly, matching your design. [14wb](#)



System Administration

- 75 The Enterprise Manager Boldly Goes Forward** by *Jeff Ash*
Live long and prosper using the expanded constellation of Enterprise Manager capabilities that take advantage of the latest desktop, browser, and mobile technologies. [14em](#)



- 89 Zero Deployment With JNLP** by *Jeff Ash*
Discover how Java Web Start allows users to launch and run applications directly from the Internet using just a browser, eliminating the need for any special manual installation on the client machines. [14jnlp](#)



- 95 Don't Put All of Your Jetty Eggs in One Context** by *Richard Stollar*
Take a close look at how to use the new Jetty.xml file for better control and greater flexibility with all of Jetty's tasks, and pave the way for better things to come. [14jetty](#)

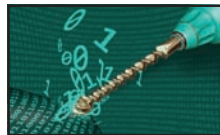


Building Blocks

- 8 Dash Boredom With the Dashboard Utility** by *Nick Decker*
Uncover the more prominent features and capabilities in the new Dashboard Utility that will add pizzazz and take the boredom out of analyzing the data in your apps. [14dashutility](#)



- 14 Ready, Set, Drill!** by *Christine Hawkins*
Drill down for the instant gratification of having data magically appear at your fingertips when and where you need it using new Drilldown and Query Definitions. [14drilldown](#)



- 30 The Magic of the Widget Wizard** by *Brian Hipple*
Abracadabra, the Widget Wizard magically generates BBj object-oriented code to create, manage, and display widgets on a desktop or a variety of mobile devices. [14widget](#)



- 43 Easier Decision Making With the Dashboard Utility** by Nick Decker
Delve into how the Dashboard Utility reduces development time and facilitates building high quality dashboards with a modest amount of code. [14dasheasy](#)



- 54 AddonSoftware's Digital Dashboard Takes Off**
by Carla Johnson and Christine Hawkins
Follow the development team as they create a sampling of ERP widgets to both pique prospects' interest and benefit VAR's own vertical development effort. [14dashaddon](#)



- 71 Makeover Your Images With BBXImage** by Nick Decker
Use this new utility to apply a wide assortment of edits to your BBImage, Java Image, server image file, or image obtained from a URL. [14bbximage](#)



- 87 You Captured My Screen!** by Ralph Lance
Easily fulfill a number of once-difficult tasks with only a few lines of code to capture screenshots and BBj windows as BBjImage objects in both GUI and BUI. [14screen](#)



- 91 Painless Payables - Anywhere!** by Kurt Williams
Save time and effort by limiting the handling of paper, eliminating the filing of paper invoices, and automating check signing with this new go-anywhere feature. [14payables](#)



Columns

- 47 Putting Your Software Through its Paces** by Brian Sherman



Learn how BASIS finds bugs faster with a Java-based professional tool that tests BBj applications automatically with a graphical user interface. [14autotest](#)



- 84 A New Day for AddonSoftware Partnerships** by Paul Yeomans



Review the new no-membership-fee Authorized Partner tier that combines product discounts and free product training; an exceptional, low risk opportunity to help Value Added Resellers expand their businesses. [14var](#)

- 106 On-Demand Enlightenment at the BASIS E-Learning Center**



by Amer Child
Sharpen your skills with the new online, on-demand, self-paced training portal, when and where you like, as it fits into your schedule. [14elearning](#)

- 109 OSAS Partner and Customer Conference** by Gale Robledo



Find out how the OSAS team is moving ahead with the latest BASIS technology and read about the BASIS highlights of the 2014 conferences. [14roadscholar](#)



- 110 BBj Logs Revisited** by Bruce Gardner



Take a fresh look at how BBj manages the logs and learn where to find them. [14trz](#)



3DTek

Innovative Search Solutions

Finding the perfect match is an art, not an accident.

Technology is constantly evolving at BASIS International, and along with it, so are your staffing needs for BBj, VPRO/5 or Barista professionals. At 3D Tek we are dedicated to helping you maintain your competitive edge by finding the right professionals for your company.

Our custom search solutions go beyond conventional recruiting, allowing us to locate, recruit, and bring employees on board who not only have the talent and skill set you need, but who share your goals and reflect your company culture.

Whether you need someone for a contract, contract-to-hire, direct hire, or a fixed price project, we can find the perfect match.

Improve your productivity and profitability with 3D Tek, your IT & Executive search and recruitment partner.



352-569-9203 or visit us at www.3dtek.com

Create Your Masterpiece

There has never been a better time to create your software application masterpiece. The BASIS ERP modules, utilities, development tools, database management, and language components are the perfect building blocks for your next great creation or for giving a facelift to your old masterpiece.

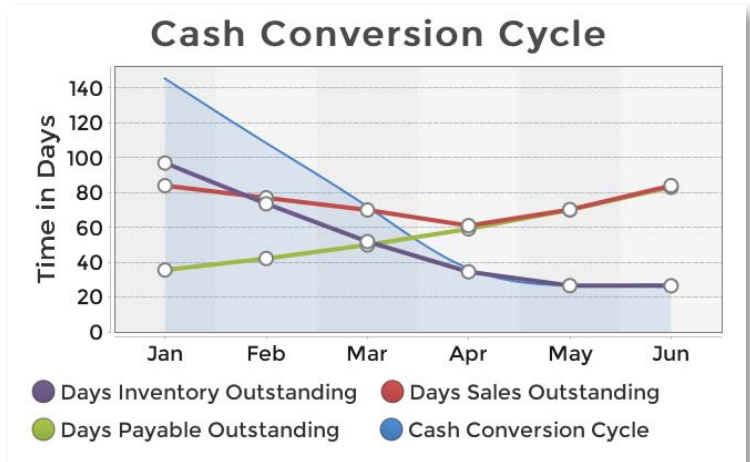
In this issue, we offer no less than four articles that cover various aspects of the new Digital Dashboard building block. Begin with the overview of the new Widget Wizard development tool (page 14) that leads you through the creation of either a single widget, a widget set, or a full dashboard of widget sets – all without writing any code – and end with the article about how the AddonSoftware team implemented a full-featured ERP dashboard (links.basis.com/14dashaddon). Speaking of AddonSoftware, “Ready, Set, Drill!” is another great article (page 8) that showcases Barista’s new drilldown and custom query functionality by showing how AddonSoftware is taking advantage of these powerful new Barista features.

Both of these new capabilities deliver on the **Information** component of the term *Information Technology*, giving the user summarized or detailed access to information via the drilldowns and presenting information in a graphical format to better facilitate good business decision making. And now, BASIS makes it easy for you to infuse your existing solution with these Business Intelligence (BI) capabilities, including on mobile devices. Try the dashboards for yourself – go to links.basis.com/addondash and imagine presenting your *Cash Conversion Cycle* data to your executives as a real-time chart on their mobile or desktop device!

Almost every article offers opportunity for you to leverage new feature and function to save you time and money while creating your masterpiece. Read on if you want to do any of the following:

- Create a mobile web app
- Duplicate data changes in near realtime to a different database
- Use the same tips and tricks that BASIS engineers use when editing with the BDT
- Implement a zero-deployment infrastructure
- Learn how others are benefiting from BASIS products and services
- Capture application screenshots programmatically
- Pass complex data structures to and from your Web Service
- Restrict access to applications served up by the Jetty web server
- Get online training on the new tools and building blocks

Discover all the new tools and learn how to better use the old ones throughout all 28 articles of the largest ever *BASIS Advantage* magazine produced especially for your edification and your delight. Then, begin the fun as you ‘create (or re-create) your masterpiece’! ■



Nico Spence
Chairman & CEO

Editor in Chief Nico Spence
nspence@basis.com

Managing Editor Susan Darling
sdarling@basis.com

Technical Editors Dr. Kevin King, Jerry Karasz
kking@basis.com, jkarasz@basis.com

Copy Editors Paul Yeomans, Sharon Wauflé
pyeomans@basis.com, swauflé@basis.com

Bling-meister Nick Decker
ndecker@basis.com

Art Director, Graphics Patricia Catlett
pcatlett@basis.com

Electronic Production Amer Child
achild@basis.com

The *BASIS International Advantage* magazine is published and distributed by BASIS International Ltd.

BASIS does not endorse any products mentioned in the *BASIS International Advantage* other than those products licensed from BASIS International Ltd.

The trademarks and registered trademarks owned by BASIS International Ltd. in the United States and other countries are listed at www.basis.com/trademarks

All other product and brand names are trademarks or registered trademarks of their respective companies.

Subscribe at links.basis.com/subscribe



BASIS International Ltd.
5901 Jefferson Street NE
Albuquerque, NM 87109-3432

Phone +1.505.345.5232
US & Canada 1.800.423.1394
International +1.505.338.4188
www.basis.com info@basis.com

Copyright BASIS International Ltd.

Stock photo credits: www.123rf.com



A DAM-EDV Picture is Worth 1,000 Woods A Logging App for Smartphones

By Patrick Schnur
European Marketing/PR

“Say ‘Cheese!’” If you are on a walk in the forest and see a truck driver taking pictures of the logs loaded on his truck with his smartphone, don’t be alarmed. This man probably does not have an overly affectionate relationship with his load. Rather, his snapshots serve a very tangible economic purpose.

Together with pilot customer Leobner Realgemeinschaft, one of the largest logging companies in Austria, BASIS customer DAM-EDV has developed a smartphone delivery note app for drivers to use whenever they transport a load of logs from the forest to a sawmill. Let’s take a look at what this process looked like before, and how the new app works.

Before: A Slow and Error-prone Process

Previously, the driver estimated the quality and quantity of the wood and manually filled in a delivery note on paper, which he handed in with his load. At the sawmill, they weighed the wood and provided their own estimate of its quality. The measuring record taken at the sawmill is important for the logging company, because the quality of the wood defines the price the logging company receives. The price can fluctuate significantly, from 40 to 100 euros per solid cubic meter. The sawmill determines the final price and eventually credits that amount to the logging company.

It could take several days until the delivery note is entered – manually, again – into the database. The logging company would receive their money only after the wood was processed in the sawmill. And every so often, there would be conflicting opinions between the logging company and the sawmill as to the quality of the wood. But what if the logging company was convinced that the wood they had delivered was worth more than they were credited by the sawmill? How could they prove the quality after all the logs had been turned into boards?

Now: More Convenience and a Quicker Turnover

DAM-EDV’s BUI app has profoundly changed this entire paperwork-heavy process. Instead of filling in a delivery note by hand, the truck driver simply types the data into his smartphone on an app running on a cloud server. The intuitive entry screen shown in **Figure 1** makes it simple for him.



Figure 1. The entry screen on the smartphone

Then he takes a couple of pictures of his cargo with his smartphone's built-in camera (**Figure 2**). The image quality with modern phones is very good – Apple, Samsung, and Sony have all performed well in tests – and the app can process up to seven photos. So when in doubt, validating the quality of the delivered wood with a photo helps to substantiate the grade the logging company claimed. Videos are possible as well, of course. “A 60-second video will be a 5 MB file that takes only about 20 seconds to transmit,” explains Werner Dam, Managing Director of DAM-EDV.



Figure 2. Driver capturing his cargo for transport using his smartphone

With a simple tap of his finger on the 'send' button, the driver transmits the delivery note, including pictures and video, via email to the server at DAM-EDV. There, the email is forwarded automatically to the freight company and the logging company. At the same time, the digitized delivery note is stored in the logging company's database.

The simplified process has advantages for everyone.

- The driver can more easily forward his data
- The logging company
 - Eliminates the time required to enter the data into their database
 - Receives credit on their account for the delivery much sooner
 - Can better estimate their financial turnover knowing at any given time how many trucks are underway with logs, and in what quality and quantity

Of course, this process greatly simplifies resource and financial planning. “Our partner's largest customer delivers up to 200 truck loads a day, which means that, beforehand, about 600 truck loads were underway without being accounted for. Now, at any given time he has the data he needs to plan his financial turnover and the deployment of his resources,” Dam explains. “The forestry enterprise benefits from the tracking of the driver's movements as well.”

Project Duration: Only Two Weeks

“We completed the web-based solution with BBJ® in about 80 man-hours over the span of about 3 months,” Dam estimates. *“And the app was our first BBJ project!”*

The pilot customer was involved with the project team the whole time, testing new program versions in practical tests and giving valuable feedback to make a very stable and functional application. The user interface is pretty basic for the time being because, as it is the first release, it was all about stable function. For the next release though, DAM-EDV will visually enhance the application with CSS (cascading style sheets) to improve the user interface and make it even more user friendly.

“We already have a number of other prospects, but it was important to me to have an application that really runs smoothly, before we offer it to other customers,” says Dam.

The developers at DAM-EDV profit directly from their experience with the BBJ BUI app since the company is planning to migrate their 10+ application packages, which are currently running in Visual PRO/5®, to BBJ.

Everyone has a Smartphone – the Hardware is Practically Free

Thanks to BUI, the app works flawlessly on any current smartphone, which virtually everyone nowadays owns. “BUI makes the modern smartphone technology available with just the ‘tap’ of a finger. For instance, we don’t need to learn an extra programming language for native apps for each smartphone manufacturer. We can also foresee adding other functions that the hardware offers. For example, we could save the GPS code along with every photo taken, so that the truck driver could prove where he picked up his cargo,” says Dam. “If a customer demands it, we’ll make that available on the spot.”

But the most important thing is the fact that the hardware comes practically for free. “Our competitors have similar solutions in the market, but they all require buying tablets and extra software.” With DAM-EDV, the app is seamlessly included in their application package ‘Rundholzanwendung’ (log application) as an optional module, and the logging company doesn’t need to invest in any hardware. Every driver can use his own device, with which he is already familiar. Training expenses are minimized to “two minutes,” as Dam reports. “Needless to say, our customers liked this aspect very much.” And with any changes in staff, there is also no extra retraining effort. The new driver receives the URL and his credentials for the app, and begins to work productively with his own smartphone almost immediately.

Easier With Assistance From BASIS

Throughout the project, DAM-EDV was in contact with the BASIS Europe team. For example, when they discovered an issue with using the iPhone's numeric keyboard, the Saarbruecken team immediately forwarded it to the development team at BASIS USA who quickly resolved the issue and released the fix in the next BBj distribution.

In addition, Andreas Timm was available for face-to-face training and for answering detailed questions remotely. Dam thinks it is a good idea to commission detail projects to BASIS Europe, when it makes sense. *"For example, BASIS Professional Services designed the module for uploading the pictures and videos from the smartphone to the server. We could have done this ourselves because the tools are very well documented. However, an experienced programmer from BASIS delivered that in a couple of hours, saving us a couple of days' work that we could use more effectively for our customers."*

Summary

DAM-EDV of Austria, together with their pilot customer Leobner Realgemeinschaft, have developed an app for the lumber industry, dramatically streamlining the data flow of the delivery of wood to sawmills and making the process more efficient for all. Following the success of this app, there is already a huge interest from other players in the very large lumber industry business in Austria.

Werner Dam is also thinking about another version of the BUI app for other customer industries. *"For instance, building material wholesalers come to mind, who could achieve efficiency gains with mobile applications as well. Taking our growing experience into account, developing such a new app would be a piece of cake for us."* ■



DAM-EDV Ges.m.b.H. in Styria, Austria, was founded in 1988. Werner Dam has been the company's managing director since inception. Today, the company serves 110+ customers from various industries such as lumber, sawmills, sand, gravel, and logistics in Austria and neighboring countries.

Customers can choose from a broad portfolio consisting of standard solutions for accounting, cost accounting, asset accounting, and human resources, which they can combine with industry-specific applications to form their own individual IT system. All DAM-EDV customers subscribe to BASIS' Software Asset Management. www.damedv.at



Read more about native apps, hybrid apps, and web apps in [How Business Apps Go Mobile](#)

BARISTA®

***Rapid
Application
Development***

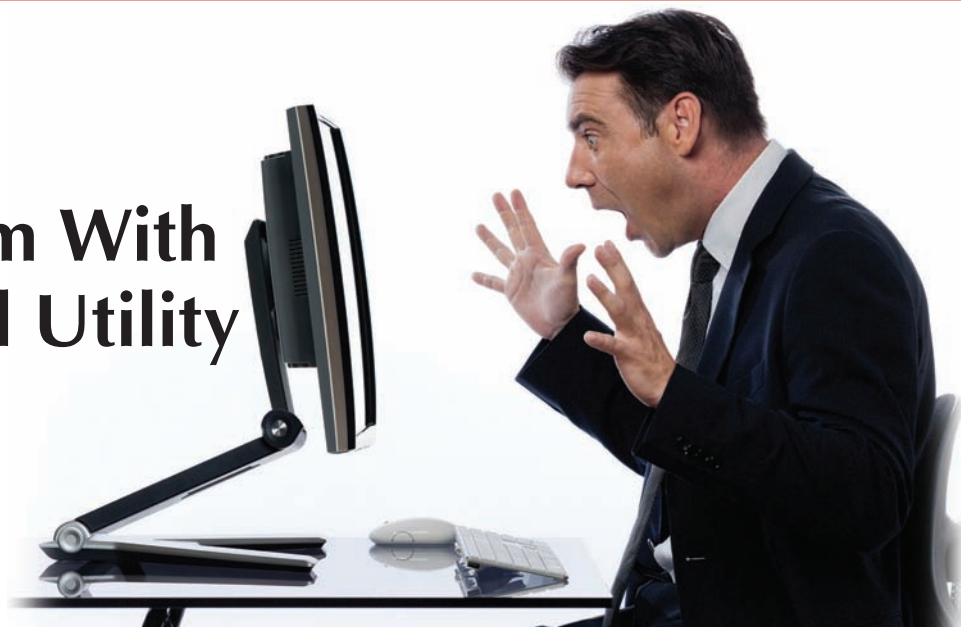
BARISTA®, the ultimate RAD tool for database-driven software projects built with BBj®, the dynamic object-oriented programming language for Java

BASIS
INTERNATIONAL
www.basis.com

Dash Boredom With the Dashboard Utility



By Nick Decker
Engineering
Supervisor



Following the release of the Dashboard Utility in BBj® 14.0, BASIS introduced this utility in a few [Java Breaks](#) and published several live demos on the [BUI Showcase](#) page. To recap, this article covers some of the Dashboard Utility's more prominent features and capabilities, and provides images and screenshots to give you a better feel for all it has to offer.

Dashboards – A Blast From the Past

At BASIS TechCon in 2007, we presented a GUIBuilder-built demonstration program called DigitalDashboard. BBjCharts were new to the language at the time, and the program's purpose was to illustrate how to utilize those charts to show the Chile Company's sales team's performance.

Fast forward several years to the release of BBj 14.0 that now comes bundled with the [Dashboard Utility](#). The underlying technology of the Dashboard Utility is the same as that of the [BBjChart](#) methods, but the differences are like night and day.

- BBjCharts are independent, low-level controls that you add to your BBj program and manage with your code.
- The Dashboard Utility is an advanced framework built upon the foundation of a Dashboard Widget, which includes multiple types of charts and also supports grids, reports, images, and HTML.
- The utility provides built-in widget management so that it will take care of categorizing, sizing, positioning, hiding or showing, and even refreshing your widgets for you.

After comparing the original TechCon demo with a couple of BUI dashboard apps running in iPads shown in **Figure 1**, the old slogan "You've come a long way, baby!" comes to mind.

Why Dashboards are so Effective

The core concept of a dashboard is to take complicated information and present it in a simple visual format. This makes the data significantly easier to understand, allowing you to grasp several performance metrics at a glance. Charts



Figure 1. The 2007 Digital Dashboard demo (left) compared to the contemporary dashboard BUI apps running on iPads (right)

are particularly effective, as they exploit our ability to correlate visually spatial locations and distances, sizes, and colors. These fuse together and leave us with a distinct notion of data relationships in a way that is immediately obvious – something rows and columns of numbers often cannot do. In so doing, owners and executives are empowered to make faster and more informed decisions to make their businesses better. **Figure 2** shows three different dashboard chart widgets that demonstrate data correlation.

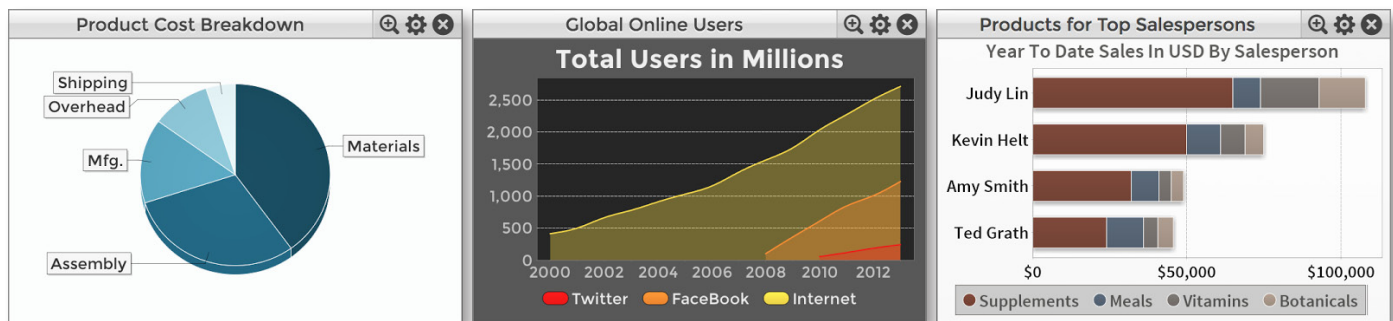


Figure 2. Dashboard widgets demonstrating data correlation in pie, line, and stacked bar charts

The pie chart shown on the left gives a clear relationship of the relative amounts a business spends on product costs based on the size of the pie slice. The line chart shown in the center visually indicates the increasing number of online users as the years go by. The slopes of the lines in the chart also convey each of their relative growth rates. The stacked bar chart shown on the right provides several points of interest. The length of the bars and the ordering of the sales reps make it simple to discern their relative performance over the past year. Looking more closely at each sales rep's bar also presents us with a breakdown of their yearly sales by product. In each one of these cases, our dashboard charts effectively present complex trend, cost, and sales information in a readily consumable format.

The Dashboard Utility's Goals

When BASIS engineers devised the Dashboard Utility, two leading tenets guided the design philosophy.

1. Developers should be able to get a dashboard with widgets running without a lot of code.
2. Widgets should be customizable, but they ought to look good by default.

How well did we do on each of those? As a testament to the former, our [Adding the New Digital Dashboard to Your App](#) Java Break video not only introduces the utility, but gives a complete walk-through of the dashboard creation process. During the video, we show the code we used to create a dashboard with a grid, bar, and pie chart that all report on different facets of sales data that was exported from a spreadsheet. The YouTube page (links.basis.com/youtube) includes a link to the final source code that is just over 30 lines, not counting the REM and empty lines that exist to aid in legibility.

The resultant dashboard shown in **Figure 3** confirms that we succeeded in our first goal. In just over a couple dozen lines of code, we have a fully functional dashboard up and running.

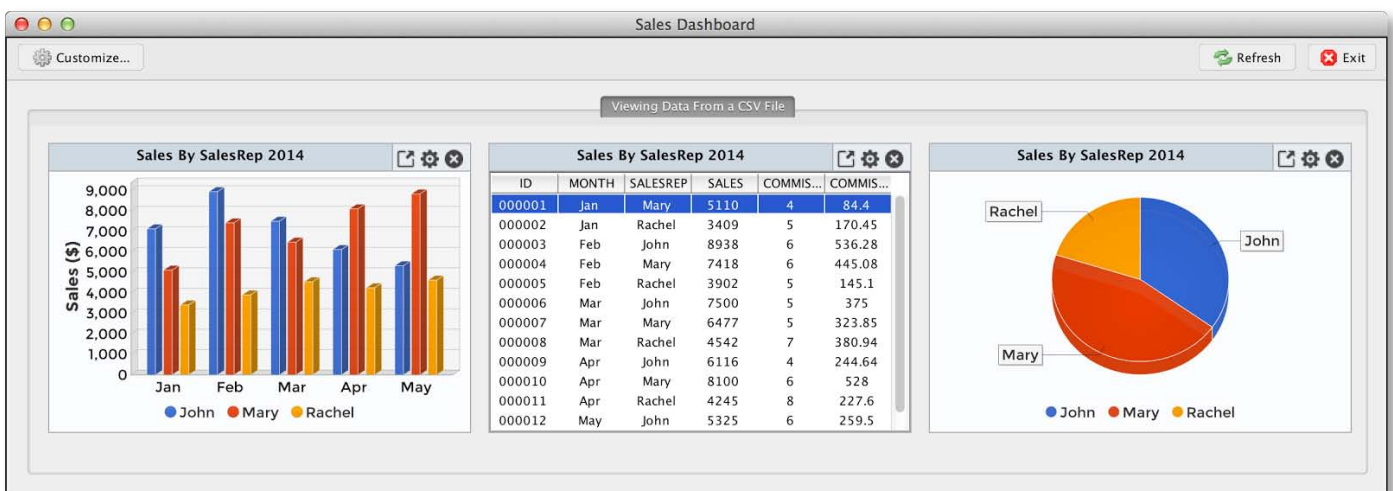


Figure 3. The dashboard program built during the Java Break

What do we mean by 'fully functional'? When you resize the window, the widgets will automatically resize and reposition themselves to maximize their use of the available window space. You can also reposition the widgets, hide and show them, pop them out to view a larger version; save out an image of the chart, email it to a colleague; refresh the widget to reflect the latest data, and more.

For our second goal, we modified dozens of chart parameters so that the default widgets look terrific without any extra work on your part. To see the difference, it's worth taking another look at the comparison of charts shown in **Figure 1**. The charts in the Dashboard

Utility are brighter, cleaner, and have improved fonts and colors that produce a more aesthetic and professional result. You still have access to scads of chart methods that offer greater control over several components including data colors and font family/size/color, but those are added enhancements instead of necessary intricacy.

The Dashboard Utility's Features

The Dashboard Utility is split between two BBJ object-oriented programs that together offer over 750 methods, so it's safe to say that the Dashboard Utility is a fairly advanced library. The list of features and capabilities is not only extensive, but has been steadily growing over the last several months since its introduction. Rather than providing an exhaustive list of every option and available method, let us take a whirlwind tour of several of the Dashboard Utility's more prominent features.

Dashboard Features

- **Dashboard Modes.** You can run a full-featured dashboard as a stand-alone program in a top-level window, or create it inside a BBJ control to embed in your application as shown in **Figure 4**. That same flexibility applies to dashboard categories (tabs) and widgets as well. This makes it possible for you to embed any portion of a dashboard into a new or existing BBJ program, or run them in a stand-alone fashion.
- **Dashboard Categories.** You can create multiple categories for a dashboard such as shown in **Figure 5**. The dashboard displays these categories in separate tabs, making it easy to provide multiple dashboard widgets without overcrowding the window or overwhelming the user.
- **Widget Size.** The dashboard resizes and repositions its widgets automatically in response to resizing the dashboard window or the browser window running in BUI. You can influence the size of the widgets by setting a minimum and maximum width, and you can also set the column and row spacing. This is done on a per-category basis, and makes it possible to have larger widgets for displaying reports and smaller widgets for displaying data grids. You can also modify the widget's size and spacing to target particular screen resolutions when running on a mobile device.
- **Widget Placement.** You can reorganize widgets in a dashboard category by simply dragging a widget via its title bar (see **Figure 6**) and dropping it into the desired position. The dashboard rearranges the other widgets automatically to make room for the newly positioned widget. The dashboard stores the position for each widget in a cookie so after you arrange the widgets, it will remember their order and display them in the same position the next time you run the dashboard.



Figure 4. A pie chart widget embedded in an AppBuilder-generated application

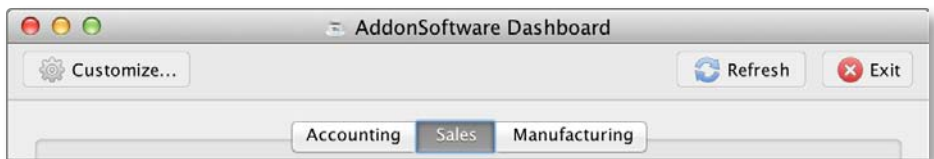


Figure 5. Examples of different categories in the AddonSoftware Dashboard

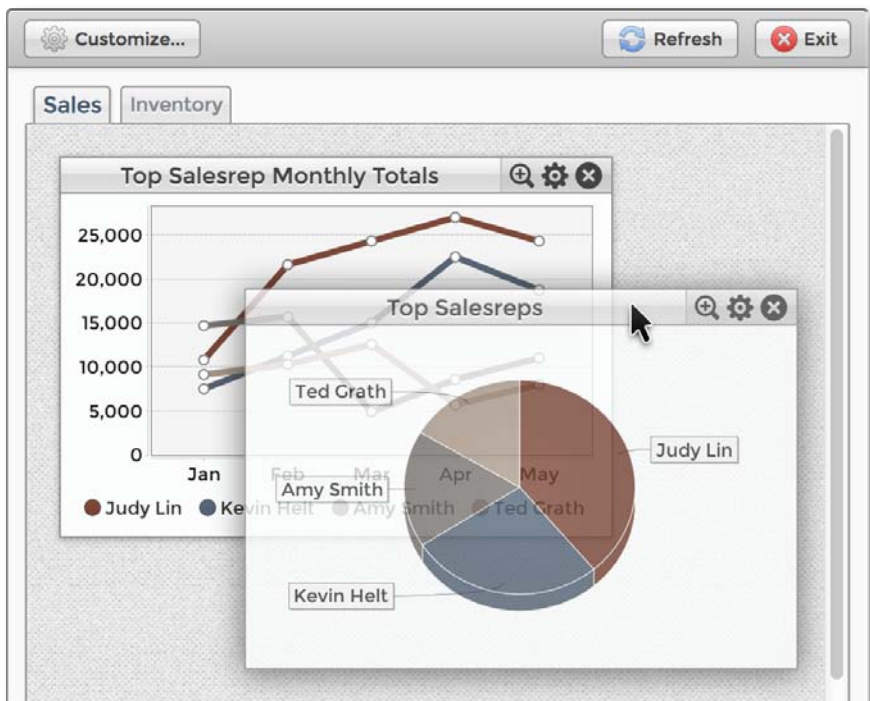


Figure 6. Dragging a widget to reposition it in the dashboard

- **Widget Control.** Dashboard widgets all have a toolbar on the top of the widget, as shown in **Figure 7**, that allows you to 'pop out' the widget, configure the widget, or close the widget. Closing a widget in the dashboard hides it so that it no longer appears in the dashboard. You can click on the dashboard's [Customize] button to see the full list of available widgets for the current dashboard category, along with a screenshot and brief description of each widget. The 'Customize' window makes it easy to add a hidden widget back to the dashboard. The dashboard also stores the visible/hidden state of the widgets in a cookie, so if you hide a widget then the dashboard will hide it the next time you run the program as well.



Figure 7. A sample widget's toolbar with controls on the right side

Widget Features

- **Widget Types.** The dashboard supports several different types of widgets, allowing you to display data in a variety of different formats. Charts are commonly used in dashboards, but you may also display grids, images, web pages/other HTML content, and even full-featured reports via BBJasper as shown in **Figure 8**. Various chart types are supported as well, including bar charts, stacked bar charts, line charts, area charts, stacked area charts, pie charts, ring charts, and more.

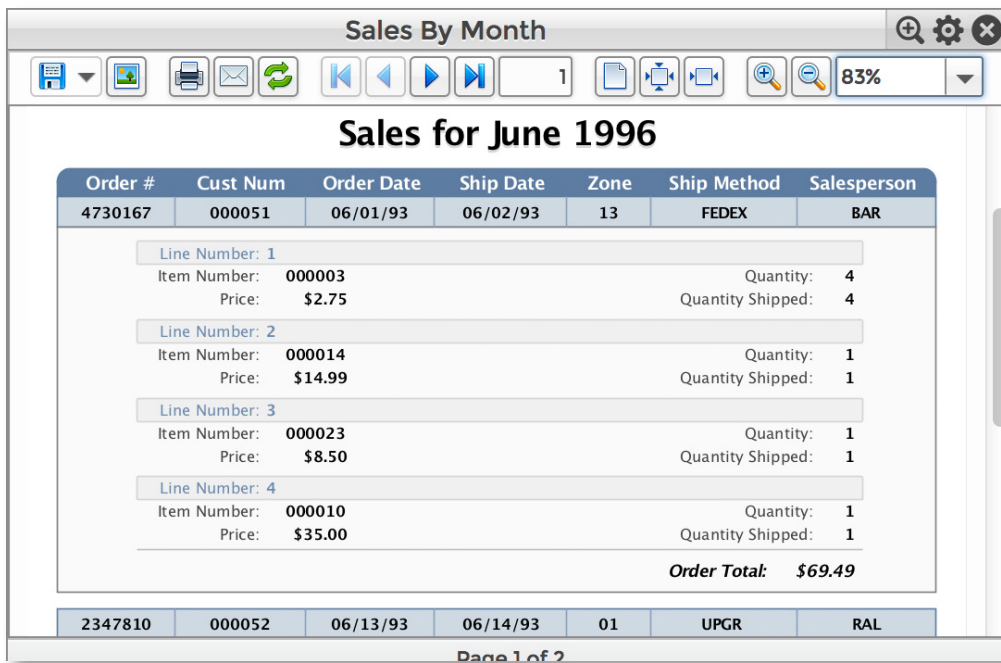


Figure 8. A JasperViewerWidget displaying a report in a dashboard widget

- **Widget Data.** You can create many dashboard widgets with an SQL connect string and query to automatically build a dataset and populate the widget. When you create a widget in this manner, it is automatically refreshable and will respond to a refresh event by requerying the database and updating the widget with the latest information.
- **Widget Refresh.** You can configure refreshable dashboard widgets programmatically to refresh themselves automatically, and users may accomplish the same via the widget's configuration menu (**Figure 9**). The widget will then update itself with new data whenever the specified refresh interval time has elapsed. You can configure the interval by providing the desired number of seconds, minutes, or hours that must elapse before the widget refreshes itself. The dashboard stores the automatic/manual refresh configuration in a cookie, so when you set a widget to refresh itself automatically it will continue to do so the next time you run the dashboard.

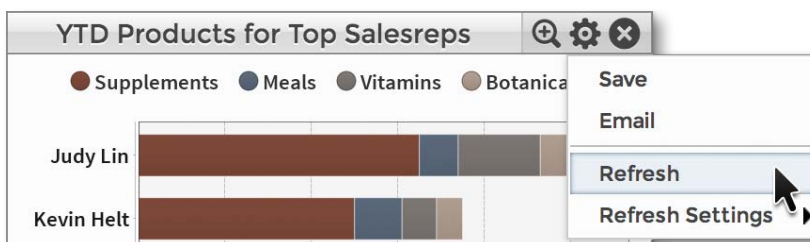


Figure 9. Manually refreshing the widget from the toolbar menu

- **Widget Filters and Links.** Dashboards widgets also support optional filters and links. Filters are displayed as dropdown lists at the top of the widget and allow the end user to modify the contents of the widget. A common use case for filters is to modify the underlying data query so that you can select values like 'Domestic', 'Overseas', etc. to change the reporting range for the widget (see **Figure 10**). Links are located on the bottom of the widget and provide an easy way to open up a web page or run a BBJ program of your choice.



Figure 10. Accessing a widget's filter to select the accounts on which to report

Utility Features

- **CSS Customizability.** The dashboard and widgets offer dozens of CSS selectors so you have complete control over the look and feel of your dashboard running in BUI. For example, a dashboard category will have the general dashboardCategoryWindow selector as well as a selector based off of the name you gave the category such as dashboardCategorySalesReports. This way you can style all of the categories and widgets at once or style each one individually with a different appearance. See the example of custom colors and CSS in **Figure 11**.
- **High Pixel Density Display.** The dashboard program and the widgets it creates take advantage of high pixel density displays when available, such as on Apple Retina devices. The fonts, charts, and even the widget and button icons were optimized to run at full resolution with more detailed graphics on a high pixel density display – both in the traditional thin client and in BUI. **Figure 12** shows a comparison of a normal display and the high density display.

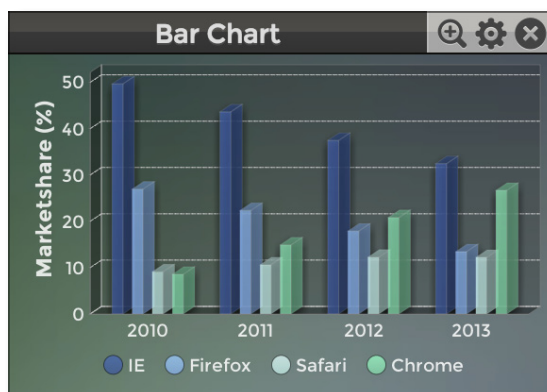


Figure 11. A chart widget with custom colors and CSS



Figure 12. The dashboard on a normal display (left) and a high density display (right) that has quadruple the resolution and increased sharpness

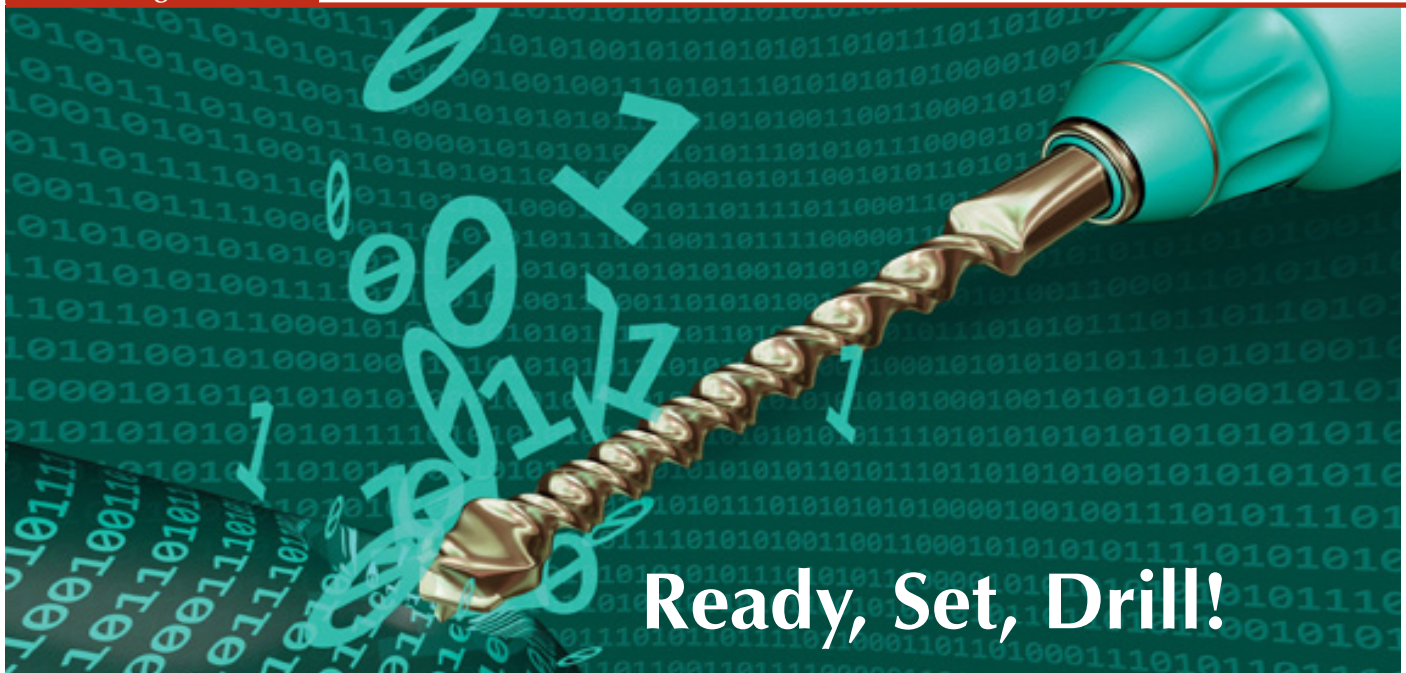
Summary

While BBJCharts have been available in the language for several years, building a dashboard using these charts has always taken quite a bit of time, effort, and code. The new BASIS Dashboard Utility makes all of that a thing of the past. By providing extensive built-in functionality and over a dozen different types of widgets, you can now get a fully functional dashboard up and running in a couple dozen lines of code. Better yet, it handles everything from sizing and positioning widgets to automatic refresh and save functionality. By taking advantage of all that the Dashboard Utility has to offer, you can equip your customers with the ability to make better, faster business decisions with a powerful and enlightening view into their data, all with very little effort on your part! ■



- Read these articles also appearing in this issue
 - [Easier Decision Making With the Dashboard Utility](#)
 - [The Magic of the Widget Wizard](#)
- Refer to online documentation
 - [Dashboard Utility Overview](#)
 - [JasperViewerWidget](#)
- Watch the Java Break [Adding the New Digital Dashboard to Your App](#) and our other [YouTube](#) videos






Who wants to spend time they don't have searching through forms and reports trying to locate information they've been asked to provide "yesterday," that they need to carry into that last-minute meeting, or that they need in order to make a management decision? Applications built with the Barista® Application Framework already provide users with a powerful set of querying tools to help them find that data and then sort, search, filter, and even export the results; now, that functionality is dramatically enhanced.

In this modern age of "Have it your way!", who wouldn't like instant gratification with data magically appearing at your fingertips when and where you need it? Now, thanks to Barista enhancements for Drilldown and Query Definitions, you can create more powerful queries than ever before, and even link queries to other queries – cascading them – or link to your own custom program! This article gives an in-depth review of these powerful tools.

Review

Queries are an integral part of Barista applications. Click the binoculars or magnifying glass on any form, run Espresso or the Document Management tasks, and you are using Barista queries. With the new enhancements, you can expand the querying capabilities of your forms even further.

Drilldown Definitions have always been a handy way to launch a query from a field on a form, because a field with a drilldown has its own toolbutton  attached. But as single-table queries, they were somewhat limited in functionality compared to queries built with the Query Definition tool. The Query Definition tool, on the other hand, has a robust feature set, including table joins, calculated columns, column totaling, auto-refresh, etc., but it hasn't been as easy to associate a custom query with a particular field on a form. In either case, the end result has been a 'single tier' query – that is, aside from the automatic hyperlinks that Barista generates for foreign key fields, there hasn't been a way to drill deeper into data in the other columns.

Well, sharpen your drill bits, because all of that has changed!

What's New

Let's start with drilldowns. Two new fields in Barista's 'Drilldown Definitions' form take away the limitations of the past (**Figure 1**). You can still specify a simple single-table query, of course, but now you also have the option to specify the name of a custom query or inquiry program to run when the user clicks that drilldown! That



By Christine Hawkins
Software Developer

Figure 1. Barista 'Drilldown Definitions' form with new 'Inquiry Program' and 'Query ID' fields

gives you the best of both worlds – you can harness all the extra features of a custom query, or even run a different form, report, or other program, and have the convenience of attaching the drilldown to a field so you can launch it via the drilldown arrow.

Custom queries have also been improved with the addition of a 'Drilldown Definition ID' field to the 'Query Column Definitions' form shown in **Figure 2**. This field lets you specify a drilldown definition for a query column, thereby creating your own hyperlink in the query grid. The potential here is huge, blowing away the 'single tier' query limitation. Think about it. You can run a form and click the drilldown arrow next to a field to launch a custom query, returning a new italicized hyperlink in that custom query that in turn refers to another drilldown definition, which may launch another custom query or custom program, and on and on.

Drilldowns in Action

Let's move from the abstract to some actual examples by examining some new drilldown functionality in AddonSoftware®. The 'Aging and Sales Summary' tab on the 'Customers' form has a drilldown on the 'Balance' field (**Figure 3**).

Balance Drilldown

Take a look at the improved 'Balance' drilldown in **Figure 4** and the columns labeled **A**, **B**, and **C**. We replaced the single-table drilldown of the past with a custom query that uses calculated fields for transactions against the invoice and the resulting balance, and sums those fields, as shown in column **A**. Barista automatically joins foreign key tables so we can see the Distribution Code and its description in column **B**. The invoice number column 'AR Inv No' noted in column **C** displays the new italicized link so from our 'Balance' query we can drill deeper and look at the transaction detail for the selected invoice (**Figure 5**).

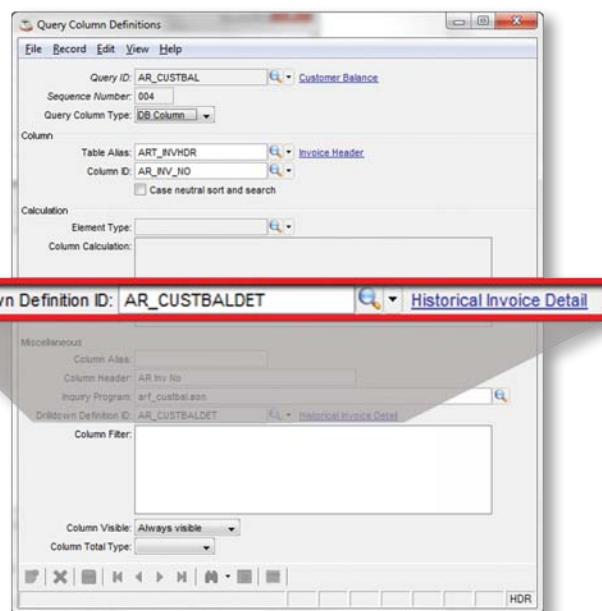


Figure 2. 'Drilldown Definition ID' field added to the 'Query Column Definitions' form

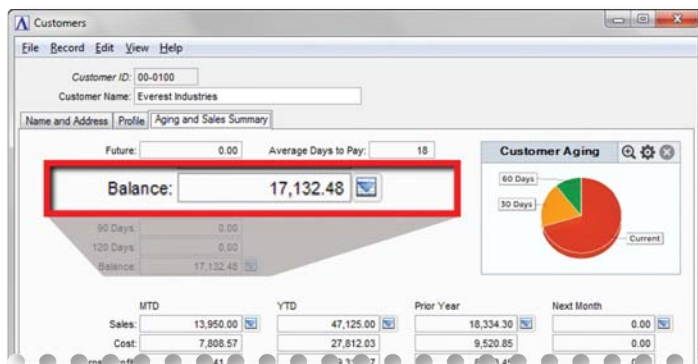


Figure 3. 'Balance' drilldown on the 'Customers' form

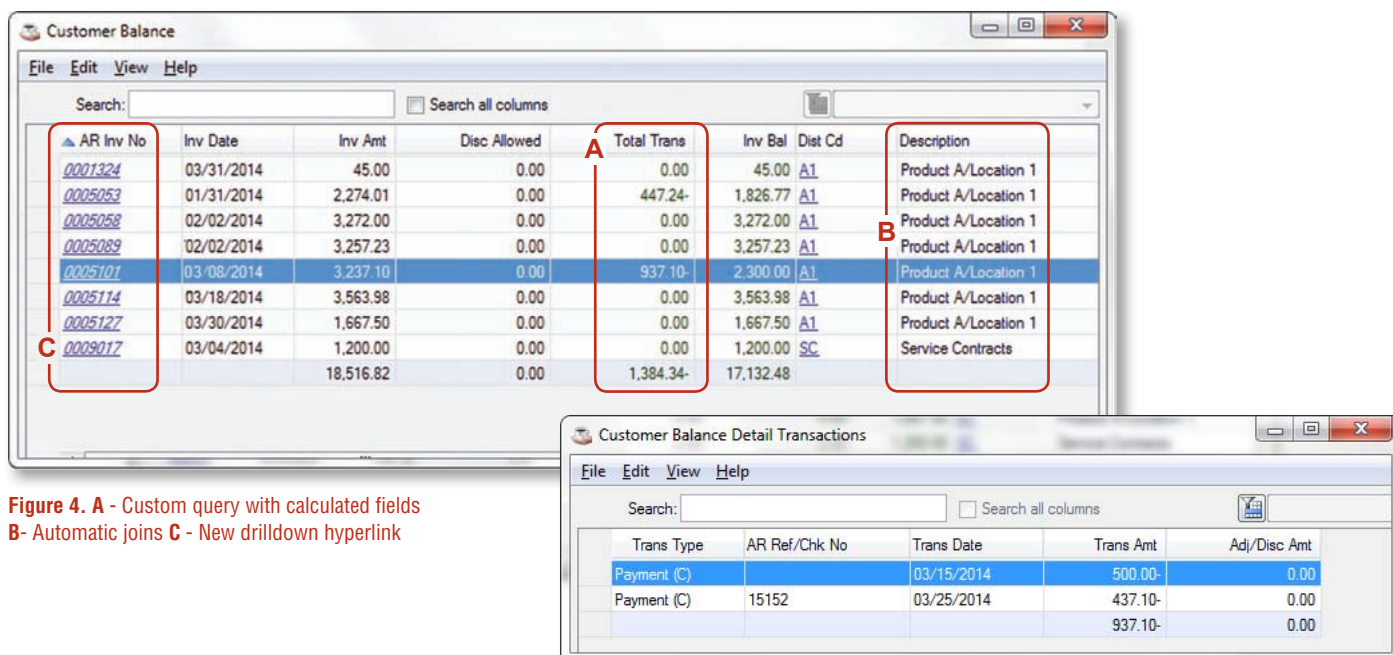


Figure 4. **A** - Custom query with calculated fields
B- Automatic joins **C** - New drilldown hyperlink

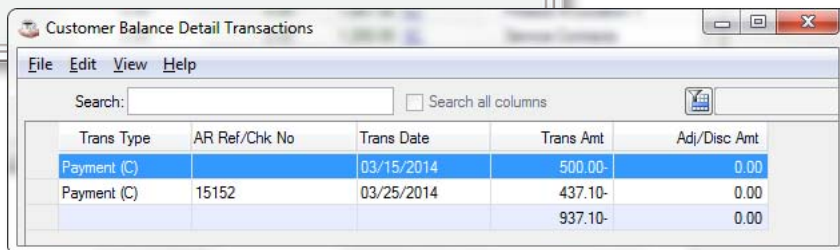
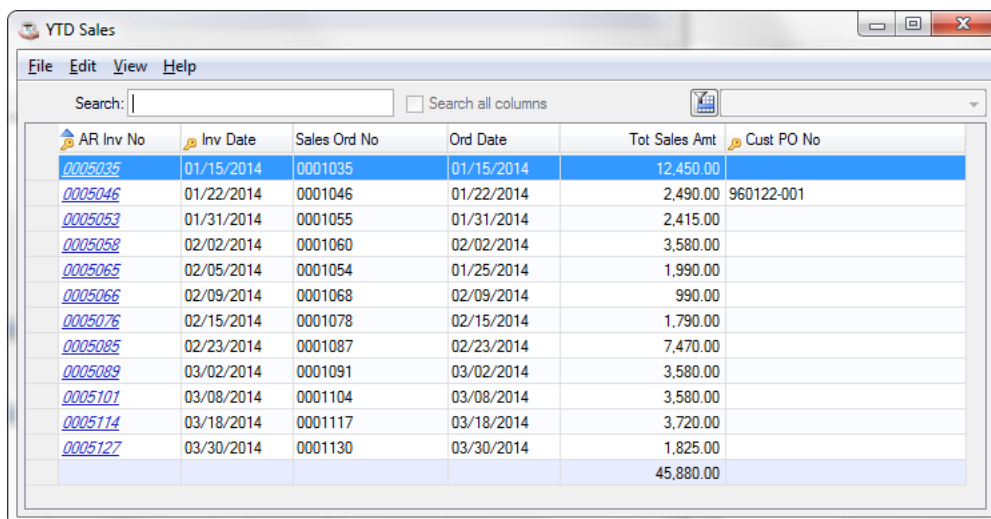


Figure 5. Custom query showing transactions against the invoice

Also appearing in **Figure 3** were four new drilldown arrows next to the sales figures. These drilldowns run custom queries as well, filtering the invoices as appropriate to show invoices from the Order Processing module that are included in the selected sales amount (**Figure 6**).

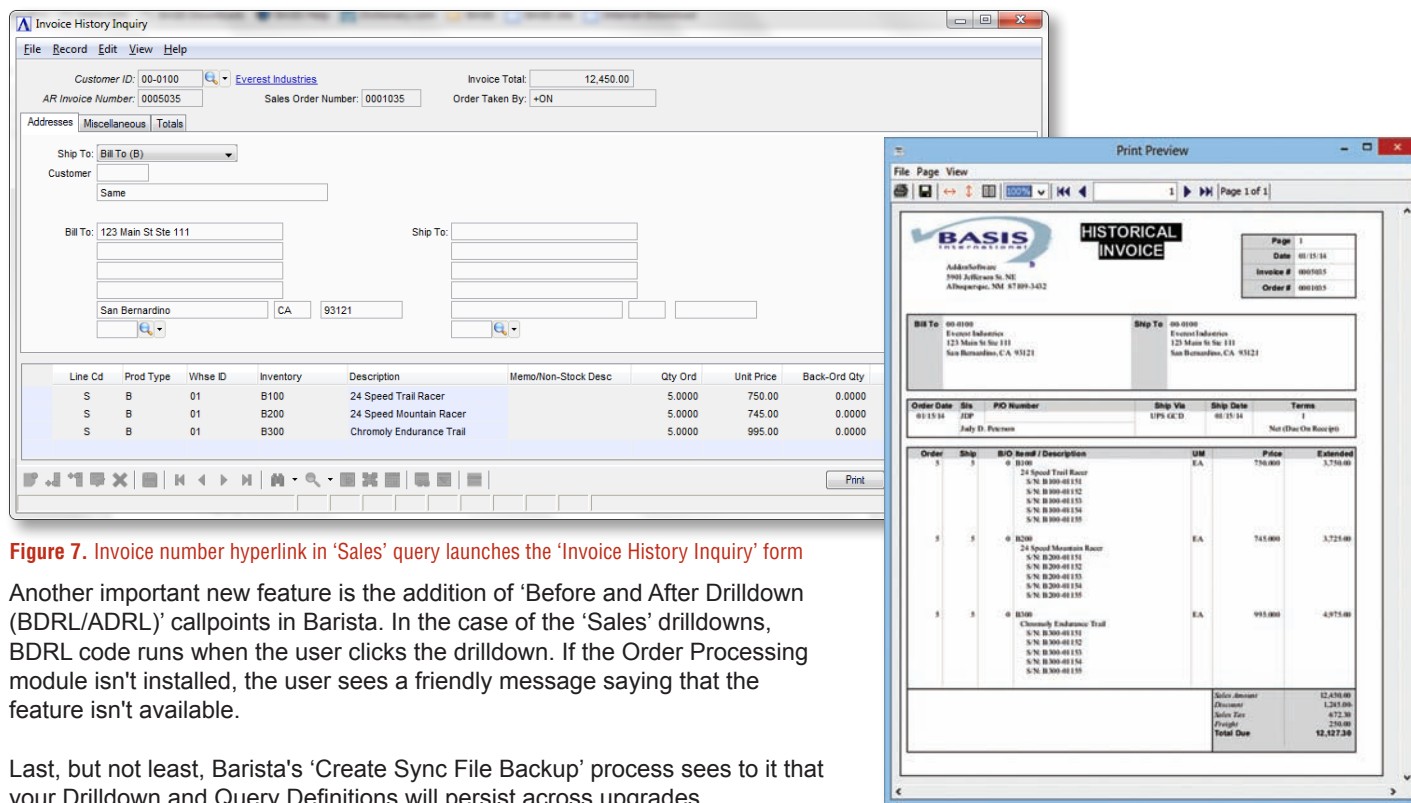


AR Inv No	Inv Date	Sales Ord No	Ord Date	Tot Sales Amt	Cust PO No
0005035	01/15/2014	0001035	01/15/2014	12,450.00	
0005046	01/22/2014	0001046	01/22/2014	2,490.00	960122-001
0005053	01/31/2014	0001055	01/31/2014	2,415.00	
0005058	02/02/2014	0001060	02/02/2014	3,580.00	
0005065	02/05/2014	0001054	01/25/2014	1,990.00	
0005066	02/09/2014	0001068	02/09/2014	990.00	
0005076	02/15/2014	0001078	02/15/2014	1,790.00	
0005085	02/23/2014	0001087	02/23/2014	7,470.00	
0005089	03/02/2014	0001091	03/02/2014	3,580.00	
0005101	03/08/2014	0001104	03/08/2014	3,580.00	
0005114	03/18/2014	0001117	03/18/2014	3,720.00	
0005127	03/30/2014	0001130	03/30/2014	1,825.00	
				45,880.00	

Figure 6. 'YTD Sales' drilldown pulls invoice information from the Order Processing module

Sales Drilldowns

As with the 'Balance' drilldown, the 'Sales' drilldowns use the new hyperlink capability on the 'AR Inv No' column. Rather than launching another query, this hyperlink runs a program that launches the 'Invoice History Inquiry' form shown in **Figure 7**, so that users can see all of the information about the invoice, print a historical invoice as shown in **Figure 8**, or even add additional comments.



Invoice History Inquiry

Customer ID: 00-0100 Everest Industries Invoice Total: 12,450.00
 AR Invoice Number: 0005035 Sales Order Number: 0001035 Order Taken By: +ON

Addresses: Miscellaneous Totals

Ship To: Bill To (B)
 Customer: Same
 Bill To: 123 Main St Ste 111 San Bernardino CA 91211
 Ship To:

Line Cd	Prod Type	Whse ID	Inventory	Description	Memo/Non-Stock Desc	Qty Ord	Unit Price	Back-Ord Qty
S	B	01	B100	24 Speed Trail Racer		5.0000	750.00	0.0000
S	B	01	B200	24 Speed Mountain Racer		5.0000	745.00	0.0000
S	B	01	B300	Chromoly Endurance Trail		5.0000	995.00	0.0000

Print Preview

HISTORICAL INVOICE

Page 1 of 1
 Date: 01/15/14
 Invoice #: 0005035
 Order #: 0001035

Bill To: 00-0100 Everest Industries 123 Main St Ste 111 San Bernardino, CA 91211
 Ship To: 00-0100 Everest Industries 123 Main St Ste 111 San Bernardino, CA 91211

Order Date	Site	PO Number	Ship Via	Ship Date	Terms
01/15/14	JEP		UPS GC/D	01/15/14	1

Order Ship B/O Item / Description UM Price Extended

5	5	B100	24 Speed Trail Racer	EA	750.000	3,750.00
			S/N: B100-00151			
			S/N: B100-00152			
			S/N: B100-00153			
			S/N: B100-00154			
			S/N: B100-00155			
5	5	B200	24 Speed Mountain Racer	EA	745.000	3,725.00
			S/N: B200-00151			
			S/N: B200-00152			
			S/N: B200-00153			
			S/N: B200-00154			
			S/N: B200-00155			
5	5	B300	Chromoly Endurance Trail	EA	995.000	4,975.00
			S/N: B300-00151			
			S/N: B300-00152			
			S/N: B300-00153			
			S/N: B300-00154			
			S/N: B300-00155			

Sales Amount: 12,450.00
 Discount: 1,267.00
 Sales Tax: 672.30
 Freight: 250.00
Total Due: 12,122.30

Figure 8. Historical invoice preview form

Last, but not least, Barista's 'Create Sync File Backup' process sees to it that your Drilldown and Query Definitions will persist across upgrades.

How it's Done

By now your mind is probably racing with all of the places in your Barista app that you'd like to use the new drilldowns. Let's review the technical details so you can get started.

Balance Drilldown

- Custom query AR_CUSTBAL queries the 'AR Invoice Header' table. The 'AR_INV_NO' column in this query carries the name of a filtering/inquiry program that is called as each header record is fetched (**arf_custba1.aon** in **Figure 2**). The filter program loops through the invoice detail for the current header record and accumulates transactions against the invoice. The accumulated transaction amount and resulting balance are placed in the query's calculated fields. The filtering program also instructs Barista to bypass any zero balance invoices.
- Drilldown definition AR_CUSTBAL (**Figure 1**) contained the Query ID AR_CUSTBAL, so this drilldown will run the custom query rather than perform a simple single-table query.
- In the Form Designer, the 'Balance' field is linked to the AR_CUSTBAL drilldown as shown in **Figure 9**.
- The AR_CUSTBAL query's 'AR_INV_NO' column also contained its own 'Drilldown Definition ID' (AR_CUSTBALDET, also shown in **Figure 2**). As a result, the invoice number column in the query grid will appear as an italicized hyperlink.
- The AR_CUSTBALDET drilldown, in turn, runs a custom query by the same name, and will show the detail transactions for the selected invoice (**Figure 10**).

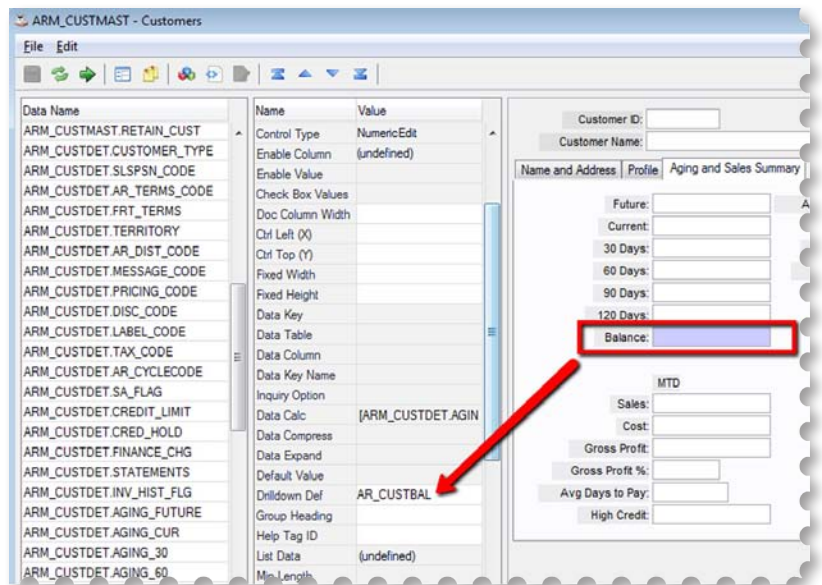


Figure 9. Use the Form Designer to link a field to a Drilldown Definition

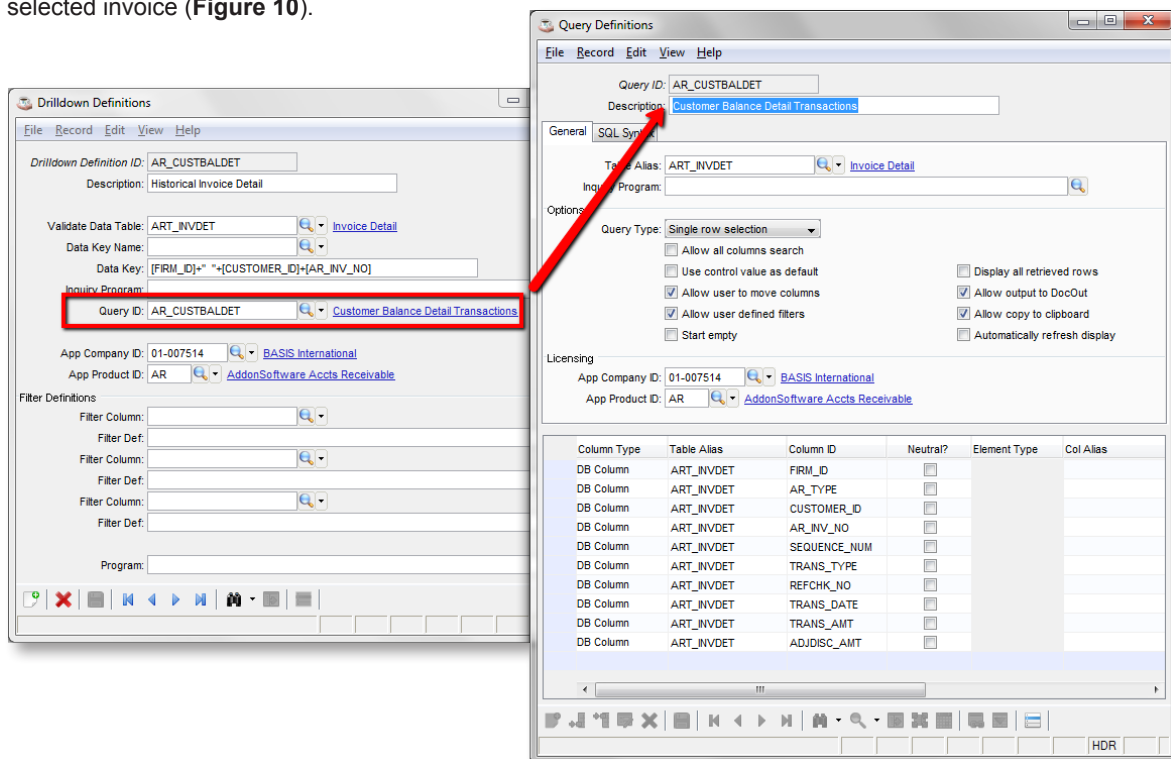


Figure 10. AR_CUSTBALDET drilldown definition and corresponding query definition

Sales Drilldowns

- The mechanics of the 'Sales' drilldowns are much the same as the 'Balance' drilldown. Each 'Sales' drilldown definition refers to a custom query, and the custom queries use additional filtering programs to select invoices appropriate to the time frame (MTD, YTD, etc.).
- Like the 'Balance' drilldown, the queries run from the 'Sales' drilldowns have a drilldown defined on the invoice number column, so it appears as an italicized hyperlink. Unlike the 'Balance' drilldown, however, this OP_HISTINV Drilldown Definition ID runs a custom program rather than another query (**Figure 11**).
- The custom program in this case (code shown in **Figure 12**) is very small, and uses Barista's `bam_run_prog.bbj` to launch the 'Invoice History Inquiry' form. Note that since Barista calls the custom program, the enter statement must always be the same as shown in the example.
- The 'Before Drilldown (BDRL)' callpoint code shown in **Figure 13** sees to it that the user is presented with a friendly message when they click one of the 'Sales' drilldowns if the Order Processing module isn't installed.

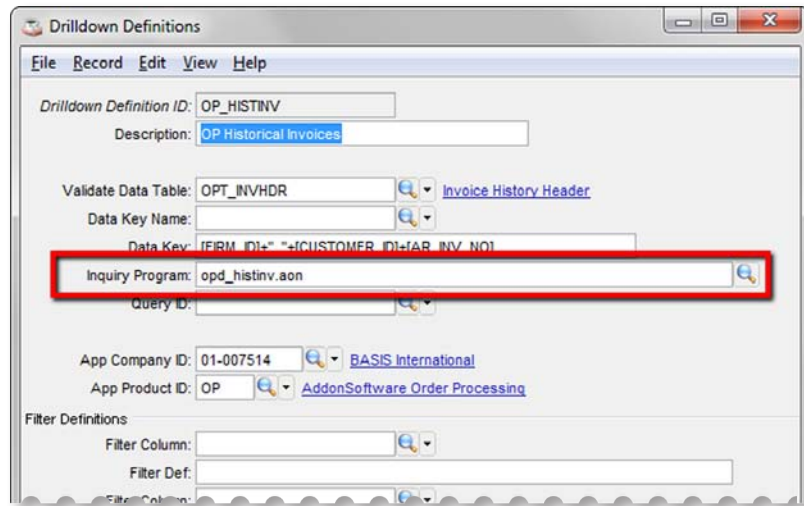


Figure 11. Drilldown Definition for the OP Sales query invoice column runs a custom program

```
rem --- arq_histInvoices.aon
rem --- launches the Historical Invoice form from AR_INVOICES custom query

enter rd_gui_dev,rdWindow!,rd_field_tvar$,rd_ddm_drill_def$,rd_drill_pfx$,rd_table_chans${all}

dim dflt_data${2,1}

call stbl("+DIR_SYP")+ "bam_run_prog.bbj",
:
:      "OPT_INVHDR",
:      stbl("+USER_ID"),
:      "",
:      rd_drill_pfx$,
:      rd_table_chans${all},
:      "",
:      dflt_data${all}
```

Figure 12. Inquiry Program used by OPT_HISTINV drilldown to launch 'Invoice History Inquiry' form

```
rem =====
check_op:rem --- this drilldown can only run if OP is installed
rem =====

if user_tpl.op_installed$ <> "Y"
  msg_id$="AD_NO_OP"
  dim msg_tokens${1}
  msg_opt$=""
  gosub disp_message
  callpoint!.setStatus("ABORT")
endif
return
```

Figure 13. Before Drilldown callpoint code in the Customer form shows message and disallows drilldown if Order Processing isn't installed

Summary

Barista queries are a powerful and ubiquitous component of the framework. Developers can extend that power by creating Drilldown and Query Definitions that provide additional access to application data in a contextual fashion, beyond where the default Barista queries can go. With these latest enhancements – Drilldowns that allow you to call a Query Definition or a custom program, and Query Definitions that let you link individual columns to a Drilldown Definition – the Barista query system has replaced the developer's hand drill with a turbocharged power drill, so you can drill down as deep as you like! ■



- Download and run the [code samples](#)



Asynchronous Triggers Modify the Copy

For a number of years, BBj® users have been able to configure triggers on data files to run a BBj program based on the particular action performed on those files. While this is a common feature in most enterprise level database management systems, standard BBj triggers have some limitations. The BBj program they execute runs inside the same Java Virtual Machine (JVM) as the Filesystem Server accessing the files so that the trigger program cannot execute on a remote installation of BBjServices or even locally in a separate JVM. Further, standard triggers must complete execution of the BBj program before the file system operation can finish, blocking that file operation until the trigger code returns. The new asynchronous trigger functionality allows triggers to fire asynchronously on a target as a result of data changes to the source file system.

One good use case for asynchronous trigger jobs is to address an often requested customer wish; *“I want to use replication to keep a copy of my data on a separate machine but I want to massage the data on the way to the replicated target.”* Of course this isn't replication because by massaging the data, you're no longer replicating the data; you're transforming the data! With asynchronous triggers, you can do just that and offload the transformation process to the target machine, leaving your production system largely unaffected.

Creating an Asynchronous Trigger Job

The best way to understand asynchronous triggers is to walk through the process of creating one. Adding an asynchronous trigger to a data file is very simple using a wizard quite similar to that used to setup replication and write auditing jobs. In fact, asynchronous triggers are built upon the replication framework, and as such they require exclusive access to the data files. Therefore, before attempting to create a replication job or an asynchronous trigger, you must first check the 'Exclusive File Access' setting in the Environment section of the BBjServices settings within Enterprise Manager. Then, restart BBjServices for this setting to take effect.

To launch the wizard, open the list of asynchronous trigger jobs by double-clicking the 'Asynchronous Trigger Jobs' node under 'File System' in the Enterprise Manager navigator. Create a new job by clicking [Add], give the job a name, and choose whether to monitor an entire database or a list of manually selected files as shown in **Figure 1**.



By Jeff Ash
Software Engineer

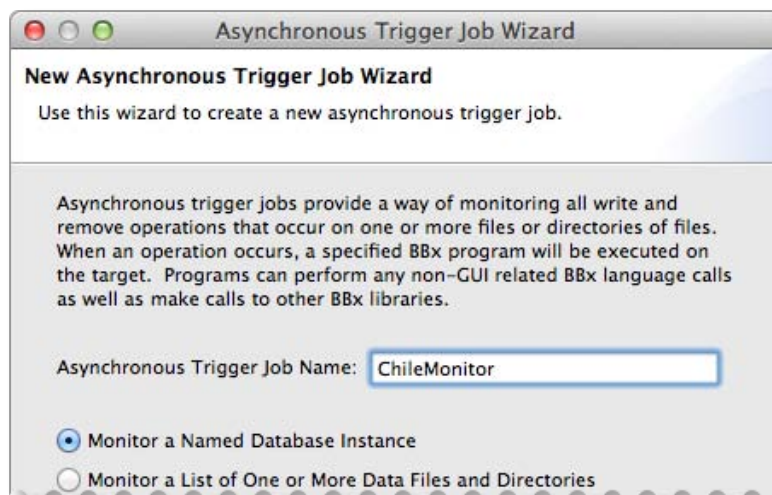


Figure 1. Choose the type of trigger job to create and name it

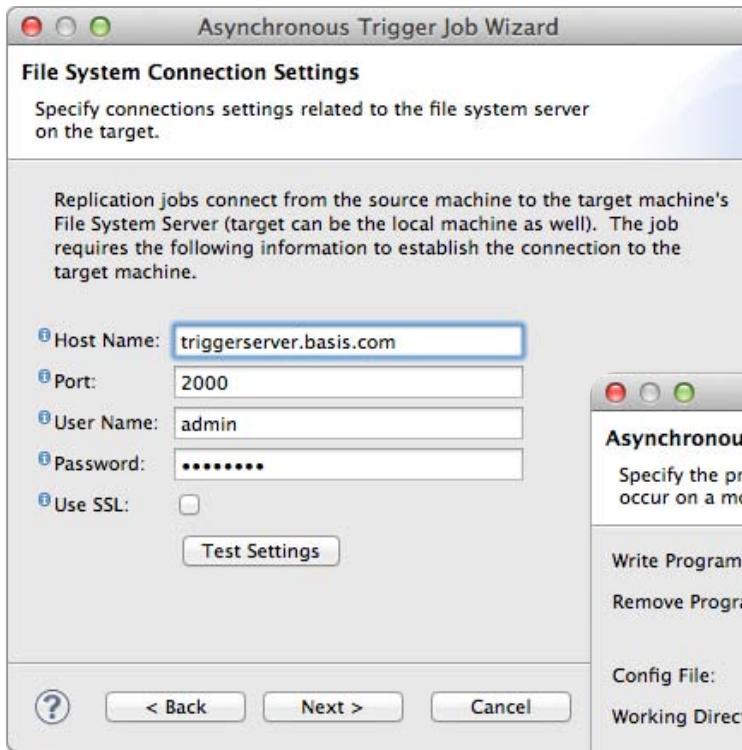


Figure 2. Specify the server to run the trigger programs

The next step is to indicate to the server where the triggers should run, which can be **localhost**, and then enter the 'User Name' and 'Password' for that server (Figure 2).

Next, configure the BBj programs to run when a write or remove operation occurs (see Figure 3). Anytime a write operation occurs on a monitored file or directory, the 'Write Program' runs. The same is true for removing a record from a file with respect to the 'Remove Program'.

Finally, configure the list of files and directories to include in the job (Figure 4), and optionally, the list to exclude from the job. Use the subsequent wizard page to specify exclusions.

Writing the Trigger Handlers

Remember the 'Write Program' and the 'Remove Program' – **writehandler.bbj** and **removehandler.bbj** – that we specified in Figure 3? They need to exist for the asynchronous trigger job to work. Fortunately, writing a trigger handler is quick and simple. These special programs support the complete functionality of BBx® with one limitation – they cannot include any user interface-related operations since the programs run inside the server, often in a headless environment.

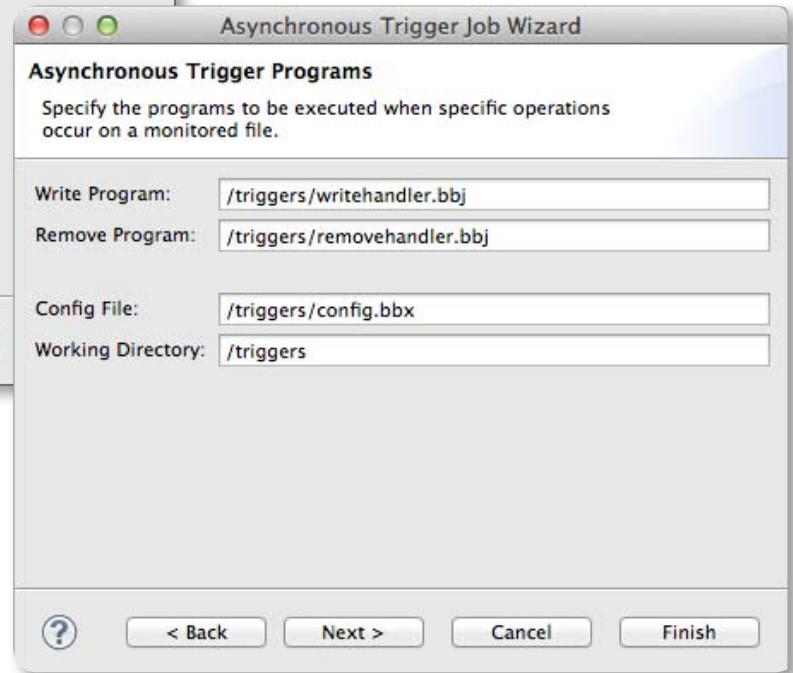


Figure 3. Specify the runtime details

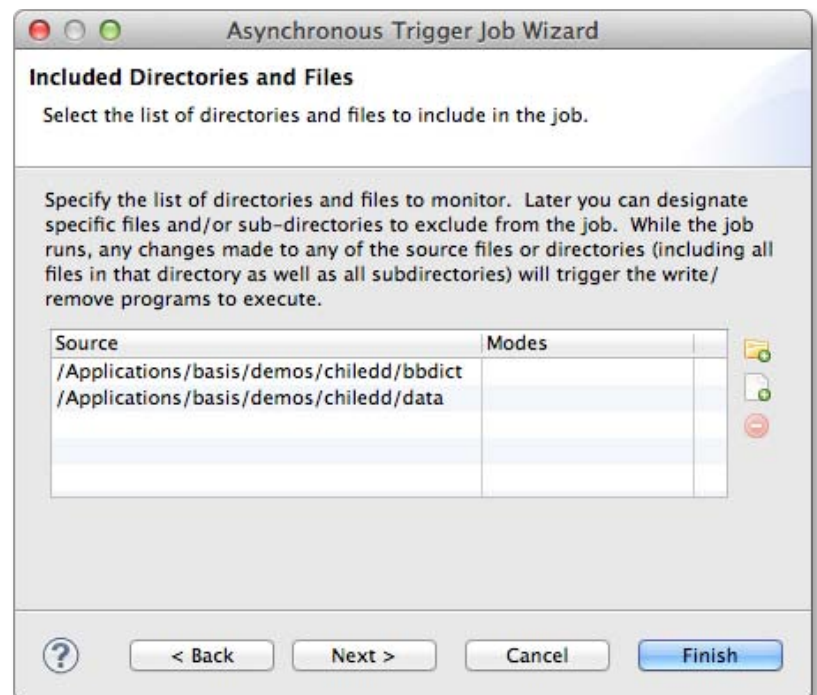


Figure 4. Specify the files and directories to monitor

Another important difference is that BBJ treats trigger handlers in a special way, making a special object available to the environment that provides valuable information about the file operation that triggered the programs execution. **Figure 5** displays a short example showing some of the special information available to a trigger handler.

Summary

Asynchronous triggers provide a seamless and efficient way to execute specific BBJ code whenever write and remove operations occur on a list of monitored directories and files – all without blocking the file operation. The beauty of this new feature is that you can configure your triggers to execute the BBJ code on any BBJServices server available on the network, remote or local. Using the BBJTriggerData object available to the trigger handlers, you have all the information necessary about the file and operation necessary to provide powerful processing. This new 15.0 feature is available for preview today so why not install it and try it out? ■

```
REM Get a Trigger object from the BBJ File System
tdata! = BBJAPI().getFileSystem().getTriggerData()

REM Get the user who performed the operation
user$ = tdata!.getUser()

REM Get the name of the file
filename$ = tdata!.getFilename()

REM If a write handler, get the record written to the file
writeBuffer$ = tdata!.getWriteBuffer()

REM If a remove handler, get the key value specified for the remove
key$ = tdata!.getKey()
```

Figure 5. An example program showing some of the information available to a trigger handler



- See [asynchronous triggers in action](#) on YouTube
- Refer to [BBJTriggerData](#) in the online documentation
- Download and run the [code sample](#)

Stimulate Your Brain!



**30-minute webinars
that make a difference!**

links.basis.com/javabreak



Have it Your Way With New BDT Preferences and Properties

Like most good software applications, the Eclipse platform supports preferences or optional settings that allow you to configure how your Eclipse workspace operates in general, as well as how your plug-ins perform. These preferences are persisted in your workspace between restarts.

The latest Business BASIC Development Tools (BDT) Eclipse CodeEditor plug-in provides a number of advanced workspace preferences and project properties settings. These options give you much finer control over how you create and manage BBJ® projects, putting you in control of your development experience. In this article, we will explore BDT's new preferences in a hands-on tutorial, after which BDT's look and feel will meet your own personal needs. Download the BDT plug-in that comes with BBJ 14.20 or higher to try it out. Then, you can have your burger and BDT "your way"!

To follow along with the rest of this article, be sure you installed both Eclipse and the Eclipse BDT plug-in per the online instructions in *Preparing Eclipse for BASIS-Provided Plug-ins* (links.basis.com/preparingeclipse).



By Kevin Hagel
Software Developer

Next, access the Eclipse preferences as instructed below. However you decide to access the preferences in your Eclipse environment, we will refer to it as Preferences > in this article.

- On Linux and Windows, go to Windows > Preferences.. menu.
- On Mac, go to Eclipse > Preferences... menu or press the Command key and the comma as shown in **Figure 1**.

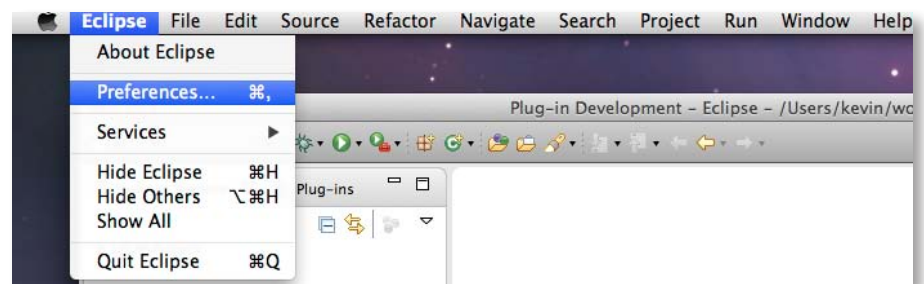


Figure 1. Accessing Preferences on the Mac

Preference settings appear listed in categories, in the left navigation pane of the Preferences window. Like other tree structures, clicking on the triangle to the left of a category expands it, and clicking on a subcategory displays its available items. Let's take a look at a few 15.0 preferences that are available in a preview release.

BDT

- BDT Dialogs

- Logging

Creation Defaults

- Source and Output File Locations

Creation Defaults/BBJ Files

- Configure Project Specific Settings

- New BBJ Files Extension

- Content Types

BBJ Compiler Options

- Filtered Resources

Errors/Warnings

- Potential Programming Problems

Content Check Preferences

Summary



BDT

Click 'BDT' in the left navigation tree to display the general setting in the 'BDT Preferences' dialog as shown in **Figure 2**.

BDT Dialogs

During your development, you may occasionally see a dialog popup warning you that something has (or hasn't) happened. Many of these dialogs include an optional checkbox 'Don't show this dialog again', which you should mark if these dialogs become annoying or you no longer need to see them. If, however, at some point in the future you decide you want those dialogs displayed next time they trigger, navigate to this window and click the [Clear] button in the 'BDT Dialogs' preference.

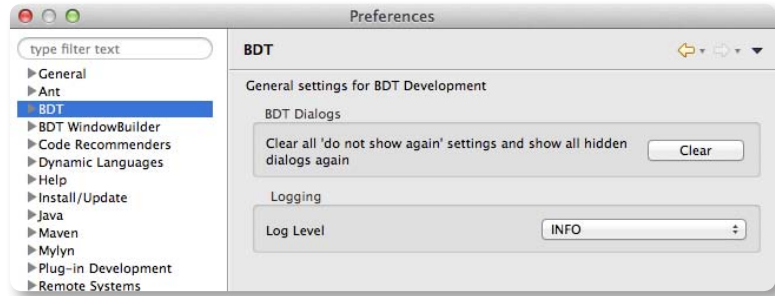


Figure 2. BDT General Settings dialog

Logging

BDT writes error and informational messages to a **bdt.log** file (located in `<your workspace directory>/metadata/.plugins/com.basis.bdt.eclipse.core/bdt.log`). If you contact BASIS Technical Support with a problem, they may ask you to set a particular Log Level value here so that specific information will appear in the log file, and then ask you to send the **bdt.log** file to help the engineers investigate your problem. The default 'Log Level' is **INFO** and you should leave it at this setting unless a BASIS tech support rep or engineer asks you to change it.

Creation Defaults

Expand BDT and click 'Creation Defaults' to display the options in **Figure 3**.

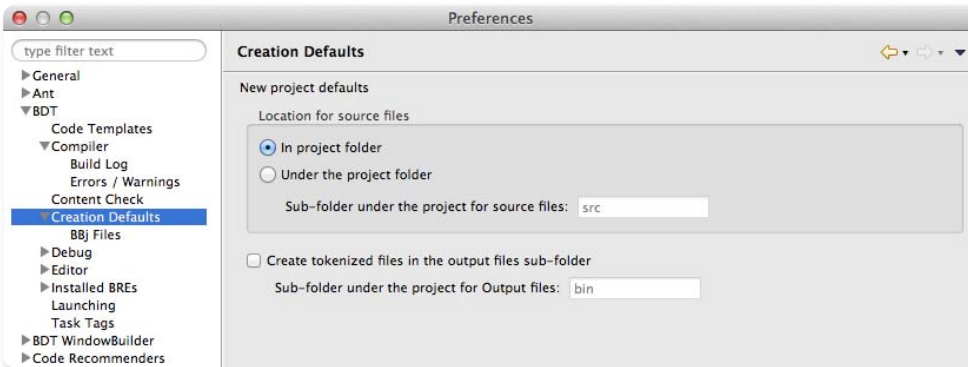


Figure 3. New project default options

Source and Output File Locations

The 'New project defaults' preferences allow you to set how BDT creates and configures new projects; these settings will not affect any existing projects.

- Location for source files
 - In project folder - By default, new projects maintain your source files in the project root folder as opposed to a subfolder.
 - Under the project folder - Choose to keep your source files in a subfolder of the project root folder by clicking 'Under the project folder' and entering a name for the sub-folder.
- Create tokenized files... - By default, BDT does not tokenize the files in your project. To have new projects' programs compiled into tokenized files, check 'Create tokenized files in the output files sub-folder'. You must also enter a name, such as **tok**, for the subfolder under the project root folder that will hold the tokenized files. Once you have done this, new BBJ projects will create tokenized versions of their source files in a **tok** folder under your project root each time you build them. To skip creating tokenized output in new BBJ projects, unmark the 'Create tokenized files...' checkbox.

Creation Defaults/BBJ Files

Click BDT > Creation Defaults > BBJ Files in the left navigation tree to display the New BBJ file defaults.

Configure Project Specific Settings

Whenever the [Configure Project Specific Settings...] appears in a 'Preferences' dialog (as it does in the BBJ Files display in **Figure 4**), it means that you can override the displayed set of preferences on a per-project basis.

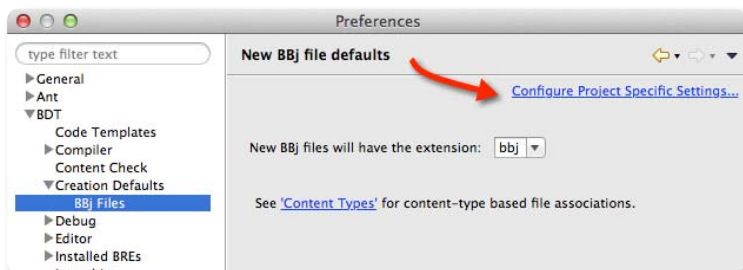


Figure 4. Accessing project-specific settings for the BBJ Files

Simply click the link and select the desired project from the list (see **Figure 5**).

Click [OK] to proceed to a filtered Properties window and set any preferences that you wish to be unique to that project (see **Figure 6**).

Since project specific preferences are optional, Eclipse requires you to mark the 'Enable project specific settings' checkbox in order to make changes. If you have set project specific settings for this project before, the box should already be marked; if not, mark it now. Once there are any preferences that are specific to this project, BDT creates a **.settings** folder in that project's folder, and places all of the settings in a **.prefs** file there. Normally, the **.settings** folder will not appear in the BDT Navigator view because it has no name preceding the extension ('settings' is seen as an extension).

You may notice that only the preferences that displayed when you clicked the link (in this case, the BBJ Files shown in **Figure 4**) are available for you to set project specific preferences. In fact, that is why the title of the window includes the text '(Filtered)'.

To view or set all of the project specific settings at one time, close all of the 'Preferences' windows and return to the 'BDT Navigator' or 'Navigator' view. Right-click the project you want to view and select 'Properties' in the menu that appears. This properties display is unfiltered (notice there is no text '(Filtered)' in the window title) and allows you to view or edit all of the project-specific preferences, even those that are not part of BDT (as shown in **Figure 7**).

If you checked out the Project Properties display, navigate back to Preferences > BDT > Creation Defaults > BBJ Files.

New BBJ Files Extension

The file extension dropdown (see **Figure 8**) allows you to set the default file extension for new BBJ files that you create. For information on creating new files, refer to the "Create a program source file in the project" section in *Creating Your First BBJ Project* at links.basis.com/creatingbbjproj.

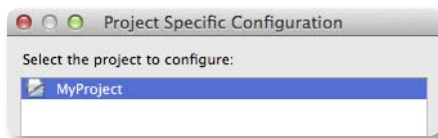


Figure 5. Selecting the project to configure

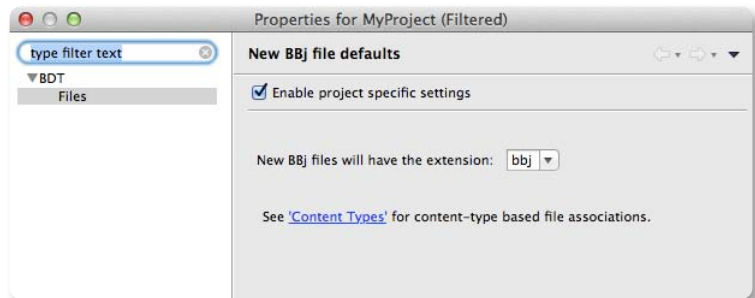


Figure 6. Filtered project-specific BBJ File options

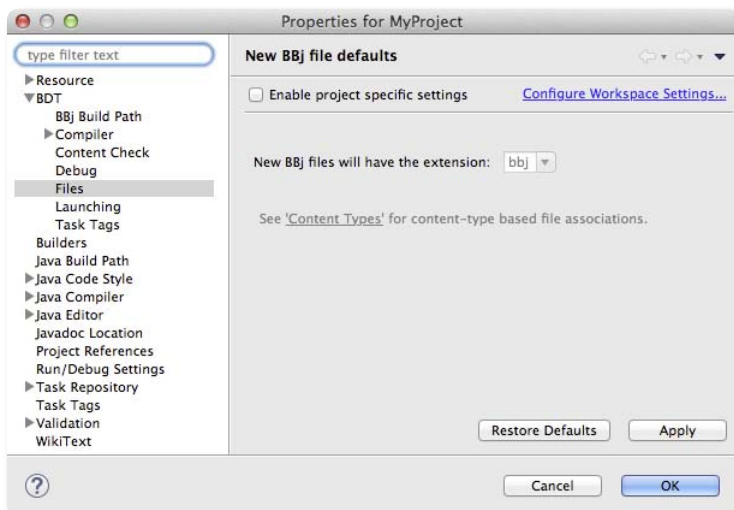


Figure 7. Project properties offering all project-specific preferences

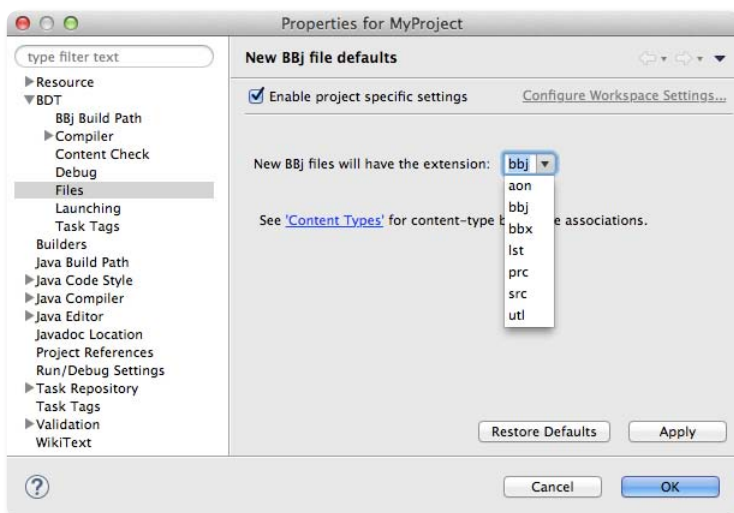


Figure 8. The list of recognized BBJ source file extensions

To choose an extension other than the default .bbj, click on your preferred selection in the dropdown list or simply type in another extension. If you type in a new extension, you must also register the extension as a BBJ content type (see below).

Content Types

In order for BDT to recognize files with your new extension as BBJ source files, you must register the extension as a 'BDT Content Type'. Click the Content Types link to jump to the Preferences > General > Appearance > Content Types display. The dialog displays a 'Content types' navigation tree; Select Text > BBJ Content Type in that tree as shown in **Figure 9**.

Click [Add...] to open a new dialog (**Figure 10**).

Type in a file extension such as
*.ext

Click [OK] and verify that your new extension now appears in the list of 'File Associations' for the 'BBJ Content Type'. Your new extension is now registered as a BBJ source file extension, and BDT's CodeEditor will now support editing, debugging, and running files with that extension as BBJ programs.

To continue on, navigate to Preferences > BDT > Compiler.

BBJ Compiler Options

Click on [Output Folders] at the bottom of the display to expand it as shown in **Figure 11**.

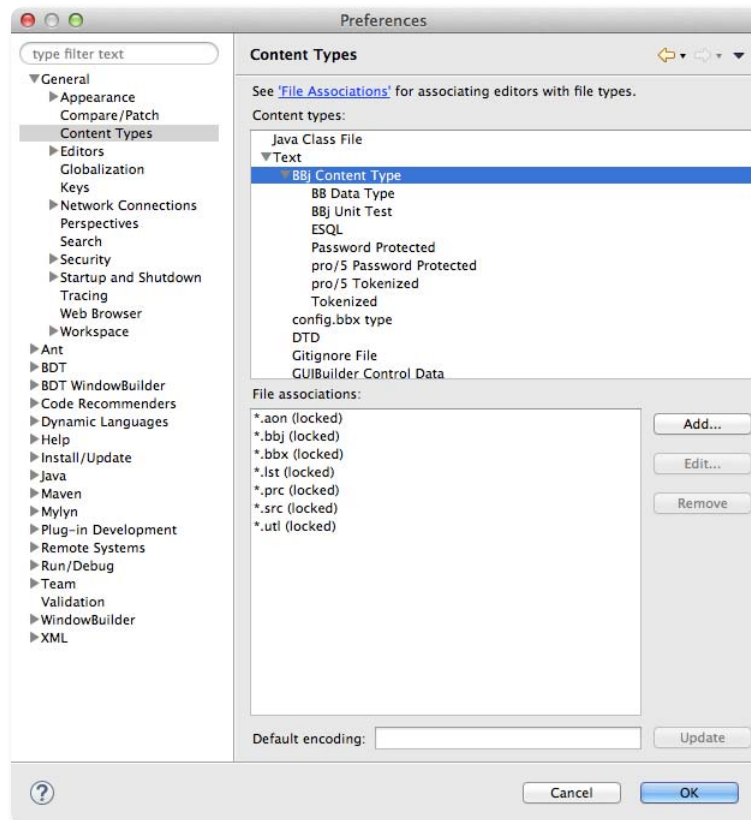


Figure 9. The files or extensions associated with the BBJ Content Type

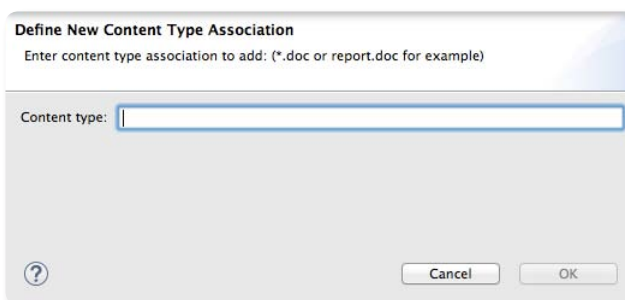


Figure 10. Associating a new extension with a BBJ content type

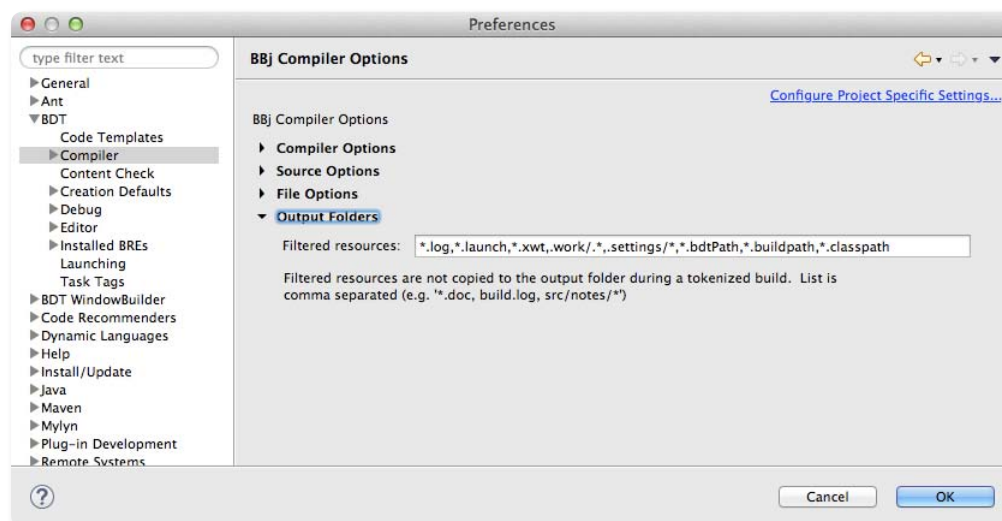


Figure 11. Filtering the files that will be copied to your Output Folder

Filtered Resources

BDT will not copy files that match these naming patterns into your output folder during a tokenized project build. By default this list includes files such as *.bdtPath, *.buildpath, and *.classpath - files which BDT and Java create for project configuration but which have no use in your output environment. You can add as many naming patterns as you like, separated by commas, using the asterisk (*) as a wildcard symbol.

To continue, navigate to Preferences > BDT > Compiler > Errors / Warnings.

Errors/Warnings

The next set of preferences we will investigate relates to warnings that may be generated when you compile a BBJ program (see **Figure 12**).

Potential Programming Problems

By default, BDT ignores a number of low priority programming issues when it detects them in a BBJ program. If you would like instead to receive a warning notification in the Problems view when BDT encounters one of these issues, select 'Warning' in the dropdown list for that issue.

As an example, if you attempt to import a BBJ library class that exists in a SAVEP form into the library, BDT will refuse. If you change the 'Import password protected' severity level from 'Ignore' to 'Warning', the next time you compile you will see something like **Figure 13** in your Problems view in Eclipse.

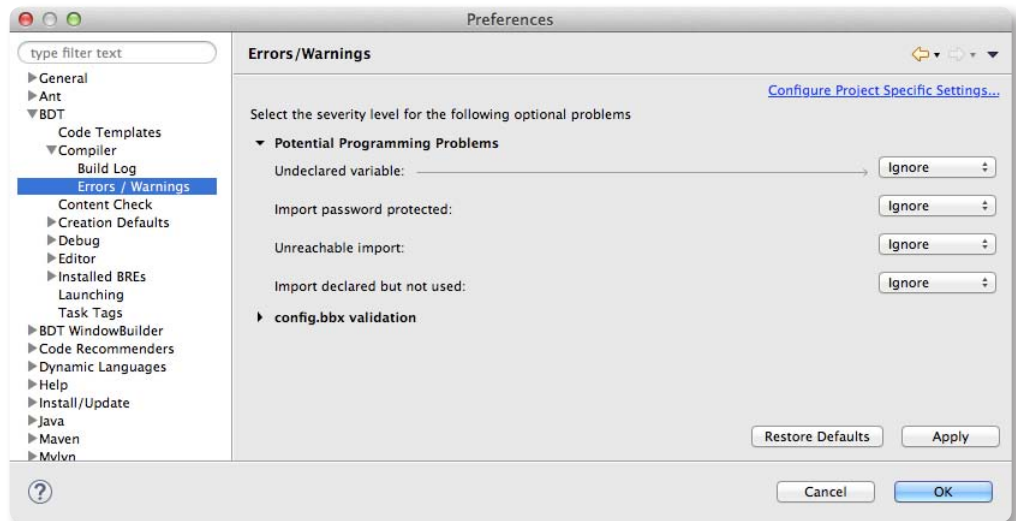


Figure 12. Notifications for potential programming problems

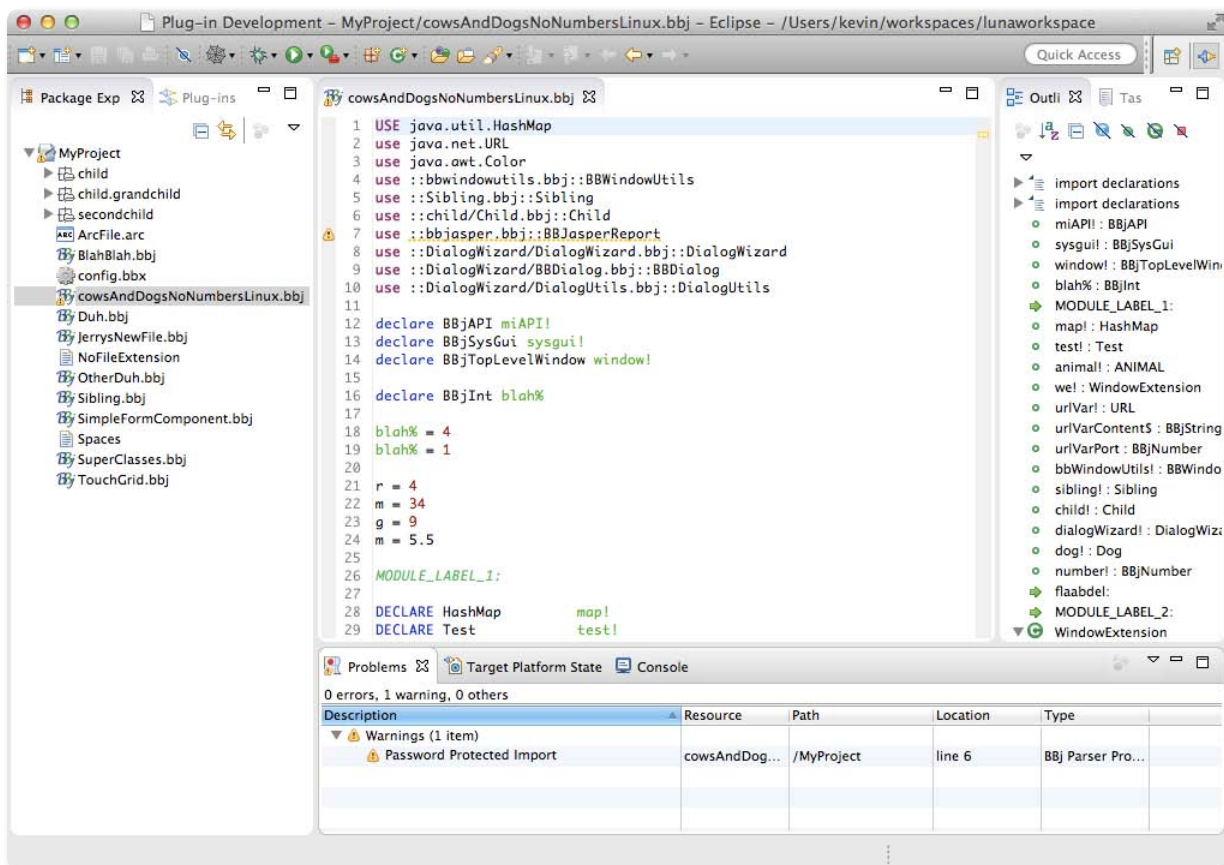


Figure 13. Password Protected import warning in the 'Problems View'

You will see similar warnings if you change the other severity levels to Warning. BASIS considers it a 'Best Practice' to always declare your variables before using them and set the 'Undeclared variable' severity level to 'Warning'.

To continue, navigate to Preferences > BDT > Content Check.

Content Check Preferences

What is Content Check? It's a time saving practice that lets BDT *handle your files correctly by displaying them with an appropriate content icon, parsing them for code completion information, and so on*. But when are content checks done and how?

Whenever events such as a project build begins, a project first opens, or when indexing occurs, Eclipse will ask BDT, "Is this a source file?" Some workspace events cause Eclipse to send every file in the project to the BDT source file validator. If BBj demands that every file have a fixed extension that correctly reflects its content, we would always be able to tell what is in a file just by looking at its file extension. But, since we can't always tell what is in a BBj file by looking at the file extension, BDT uses [Apache Tika](#) to determine each file's content type. For Tika to determine a file's content type, it must open the file and look for 'magic bytes' and other header information which might be present there. Although individual files can be opened and closed fairly quickly, when your project has hundreds or thousands of files it may take a significant amount of time to content check all of your files.

BDT is only concerned with BBj source files, tokenized files, SAVEP files, BBj data files, and so on. We call these 'BBj-interesting' files. Among other things, BBj-interesting files always copy into the output folder during tokenized builds.

Examine the 'Exclusion Naming Patterns' list in the 'Content Check Preferences' display shown in **Figure 14**.

One way to stop BDT from opening files to validate their content is to add their name to an exclusion list of naming patterns: for example, *.txt, *.xml, *.htm*. Files in the exclusion list of naming patterns are always treated as non-BBj-interesting, so BDT will not even send them to Tika for content type detection. The naming patterns supported in the exclusion list follow file search wildcard rules you may be familiar with when searching for files on your operating system.

Exclusion isn't the only interesting thing here. There is also a contrasting 'Inclusion Naming Patterns' list. So why have both? Let's say that *.txt is in the exclusion list, but you have files in your project with the .txt file extension that actually are BBj source files. If they follow a specific naming pattern, say for example **MyFile_*.txt**, you can add that to the inclusion patterns list and still see those files as BBj-interesting while excluding all other *.txt files.

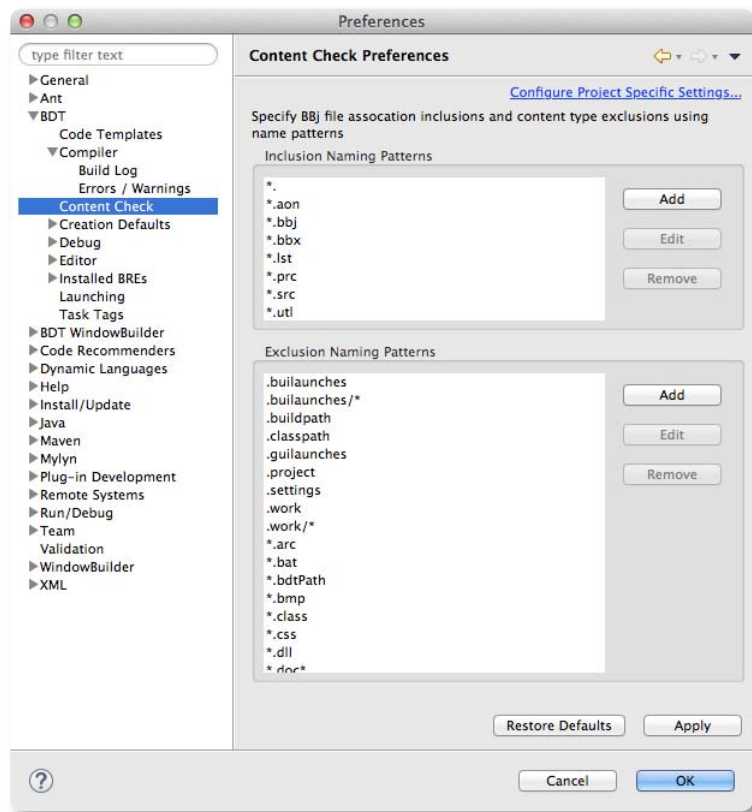


Figure 14. Exclusion Naming Patterns in Content Check Preferences

Figure 15 is a screenshot of a **MyFile_1.txt** file in an editor window before adding this naming pattern to the inclusion list:

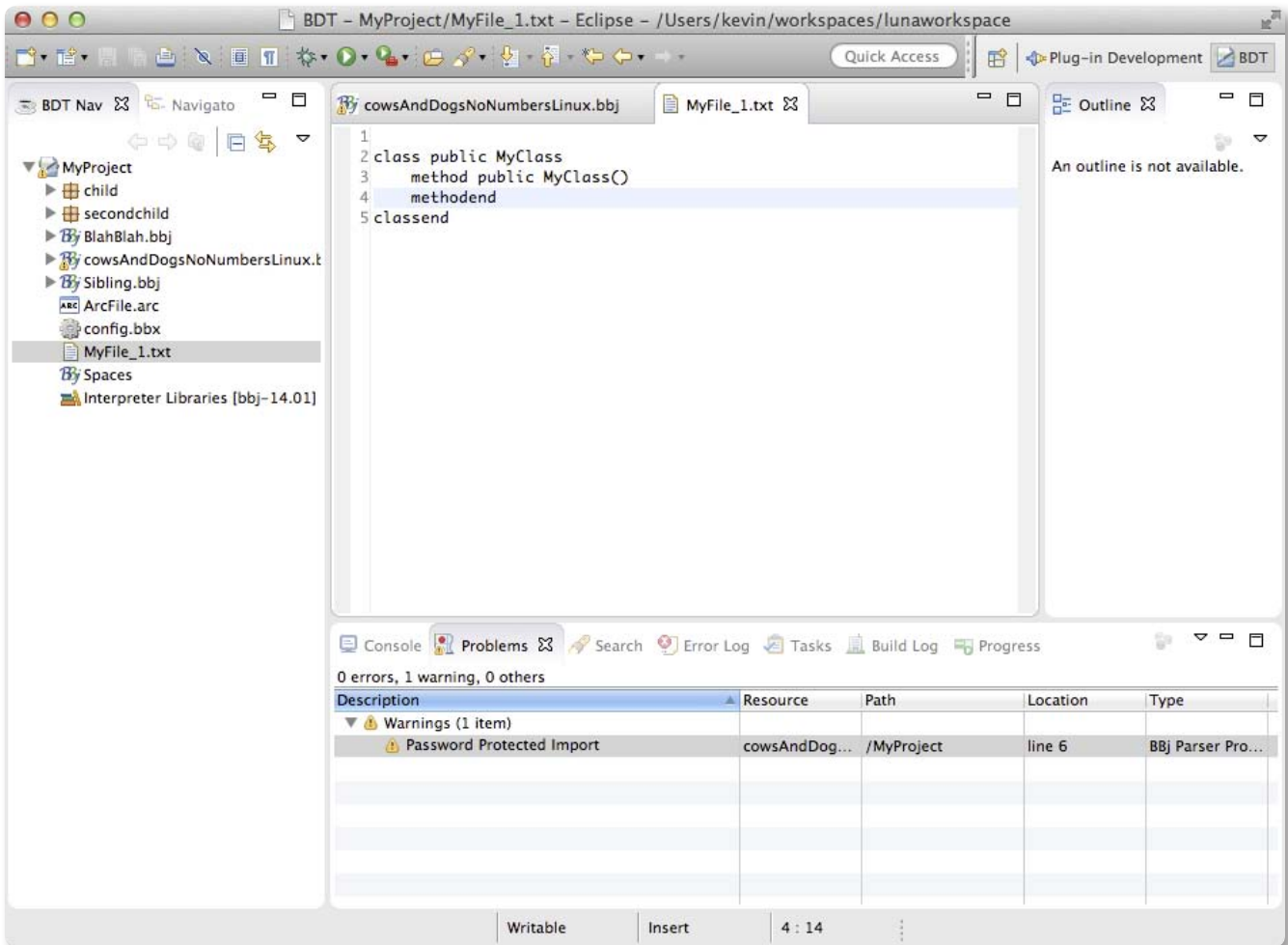


Figure 15. Workspace before adding an entry to the inclusion list

Notice in the BDT Navigator's tree control that **MyFile_1.txt** has a text file icon. Also, notice that Eclipse opened the text editor to edit the file (the icon on the 'Editor' tab is a text file icon, and there is no syntax coloring in the text). Back in the Preferences > BDT > Content Check display, click on the [Add] button next to the 'Inclusion Naming Patterns' area shown in **Figure 14**. Enter the value **MyFile_*.txt**, and click [OK]. Examine the file **MyFile_1.txt** in the 'BDT Navigator' and notice that it now shows a BBJ content type icon. **MyFile_1.txt** now appears as BBJ-interesting, as a BBJ source file.



Close the text editor window in **MyFile_1.txt**, and double-click on **MyFile_1.txt** in the BDT Navigator. This opens a new editor window. Notice that now the BBJ editor opens the file in a BBJ code editor, not the text editor (the icon on the tab is a BBJ file icon, and there is now syntax coloring in the text) as shown in **Figure 16**.

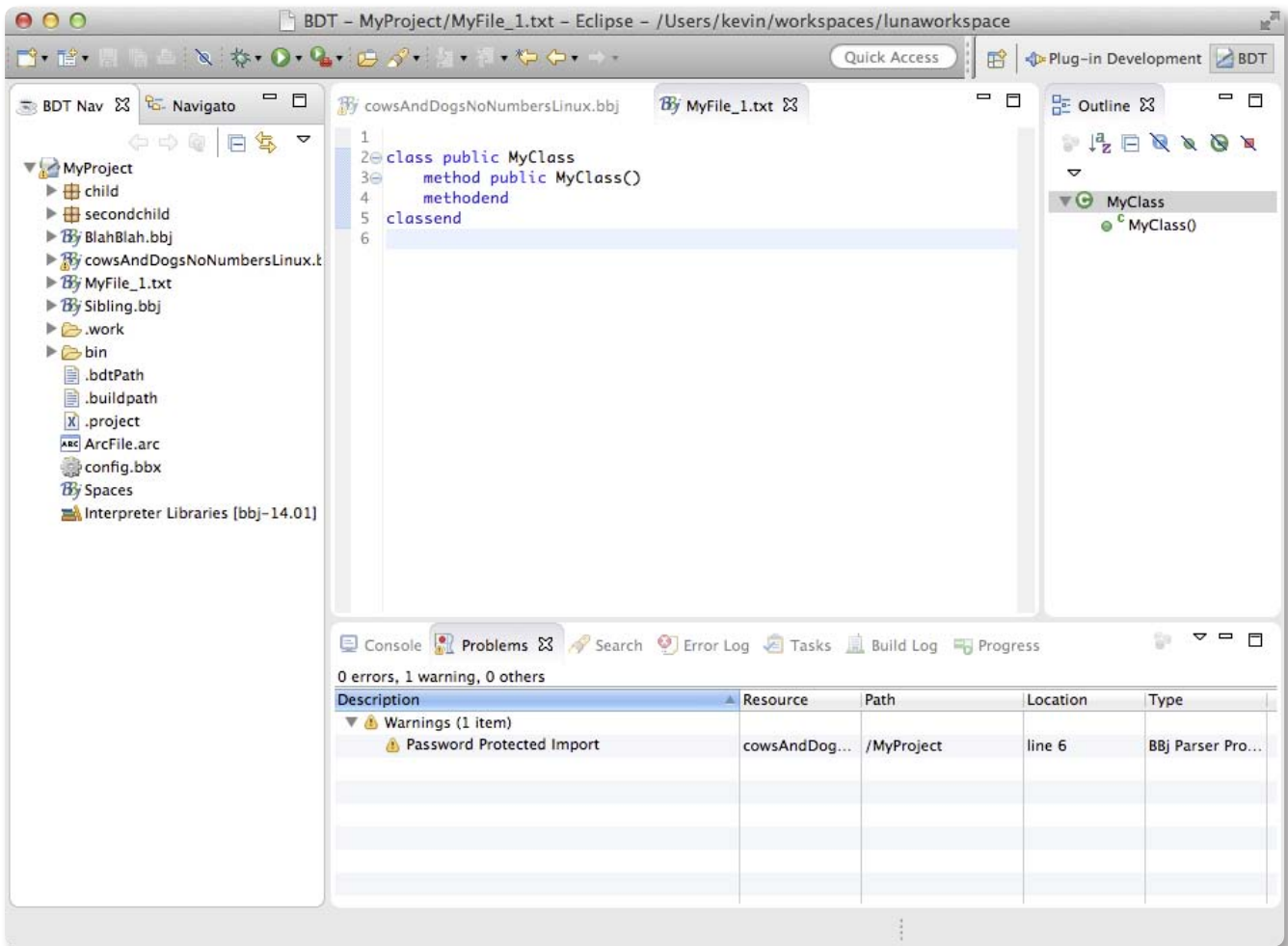


Figure 16. `MyFile_1.txt` being edited with a BBJ Content Check inclusion preference set

Summary

The BDT Eclipse CodeEditor plug-in is a very powerful and configurable tool. We have covered a number of BDT preferences in this article, but there are still quite a few that we just couldn't fit in. Please check out the online documentation for more information. If you are already familiar with Eclipse's Java Development Tools (JDT), you will find that much of BDT is already familiar to you. Either way, BDT preferences can make you much more productive, and you can make it work, look, and feel 'your way'! ■



- For instructions on installing Eclipse and the BDT plug-in, refer to [Preparing Eclipse for BASIS-Provided Plug-ins](#) in the online Help
- Read [BDT Tips for Less Pain and More Gain](#) in this issue
- Review [Creating Your First BBJ Project](#)
- Visit [Apache Tika](#)





By Brian Hipple
Quality Assurance
Supervisor

The Magic of the Widget Wizard

In BBJ® 15.0 and higher, you have the ability to create widgets – one widget, a widget set, or a widget dashboard. But what is a widget? Not to be confused with a “whatchamacallit” or a “thingamabob,” a widget is a small window that graphically displays some aspect of your data. You define the data it will display and set an optional refresh rate to make its display dynamic. So how do you create widgets without writing any code? Use magic – the new Widget Wizard!

The **Widget Wizard** is a BASIS development tool that generates BBJ object-oriented (OO) code to create, manage, and display widgets. This easy-to-use WYSIWYG utility allows you to specify the attributes and data used to generate Javadoc-commented source code. It generates the code needed to create a dashboard of widgets or to embed your widgets in controls or windows, and to run them in a browser or in a thin client without needing to write a single line of code. This allows you to use your widgets on a desktop or a variety of mobile devices. The Widget Wizard not only stores the code it generates for you, it also displays the few lines of code to cut and paste into your own program that are necessary to invoke the generated code. Because the Widget Wizard is a BBJ program, it runs on all supported BBJ platforms and can be run in GUI or BUI (browser user interface) mode.

Why Widget Wizard?

So why do you need to use the Widget Wizard? After all, it only takes a limited number of lines of code to create a widget, widget set, or a widget dashboard as shown in **Figure 1**. There can be a number of reasons.

- You may have only seen how widgets work in BASIS products, demos, or Advantage articles, but not have any personal experience with writing code.
- You might not be sure where to begin and all you want is to see a good example.
- Maybe you just want to create widgets and add them to your own application without writing any code at all.
- Perhaps you want to see the differences between the widget, widget set, and widget dashboard using your own application data.
- You may not be familiar with the BBJ OO code necessary to create and maintain widgets. The generated code can be inspected and manipulated until you achieve the desired knowledge and behavior.
- You might be wondering which widget would best display your application data.
- Even though you know your data, you may not be sure what SQL or recordset information is necessary to fill the widget with data, and you want to try out a few options.

Using the Widget Wizard to learn, to experiment with the various option combinations, or to accelerate your development will save time that translates into less development cost. In no more than eight simple steps, the Widget Wizard asks all the right questions necessary to create your widgets. Let's walk through an example.

Choosing What to Build

In Step 1 of the Widget Wizard, select what you want the Wizard to build. The choices are a 'Widget', 'Widget Set', or 'Widget Dashboard' as shown in **Figure 1**. Each choice shows an image of how it might look when you are finished.



Figure 1. Widget creation choices

The simplest of the types is the 'Widget' that creates a single widget. For example, you may want to show information for a particular customer or account in a single pie chart. To create and display multiple widgets, choose a 'Widget Set'. This type is useful for displaying multiple charts or widgets depicting a graphical representation of datasets' key to the business in a single window. Lastly, the Widget Dashboard creates and displays multiple widget sets organized by category. A category is simply a logical grouping of widget sets that a tab control visually manages. AddonSoftware® by Barista® uses a widget dashboard called the "Digital Dashboard" (Figure 2) to create and display widgets in Accounting, Sales, and Manufacturing categories.

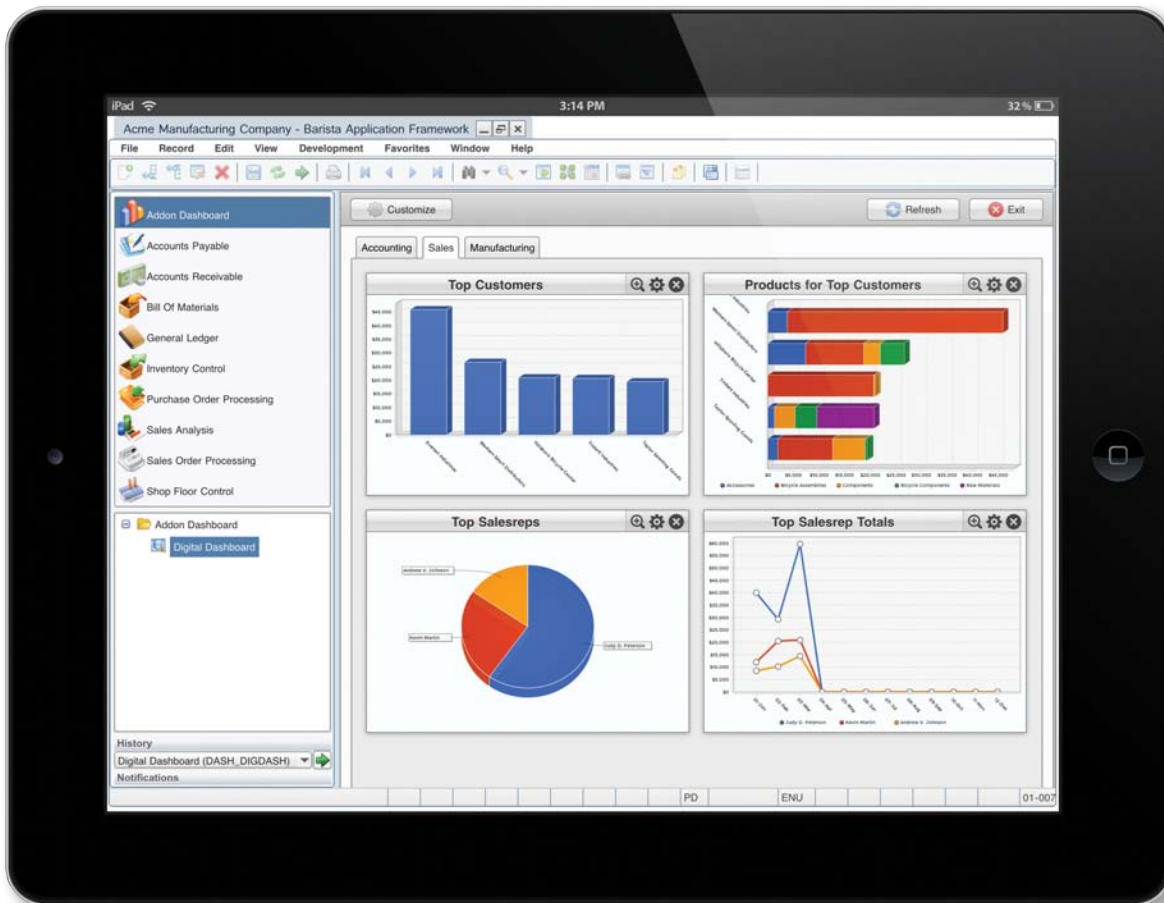


Figure 2. The Sales category in the AddonSoftware Digital Dashboard

Selecting a Widget Container

Step 2, shown in Figure 3, offers a choice of housing the widgets in either a window or a control.

Again, images provide a visual indication of what the build item might look like when selecting the associated option. The 'Window' option creates a [BBjTopLevelWindow](#) that is shown to the user in modal fashion. Program control will not return to the calling application until the window is closed by the user. The 'Control' option creates a [BBjChildWindow](#) that will embed the widgets into an existing [BBjWindow](#). This option provides the caller with control over visibility of the widgets and the application program flow.

Choosing a Widget Type

There are thirteen (yes, thirteen!) different types of widgets that the Widget Wizard can create. The widget type can range

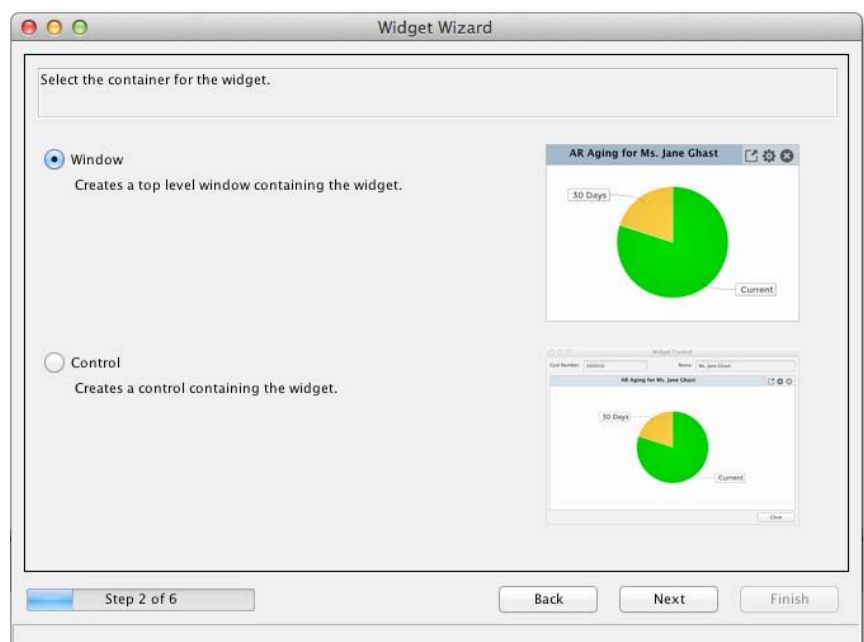


Figure 3. Select the container for the item to build

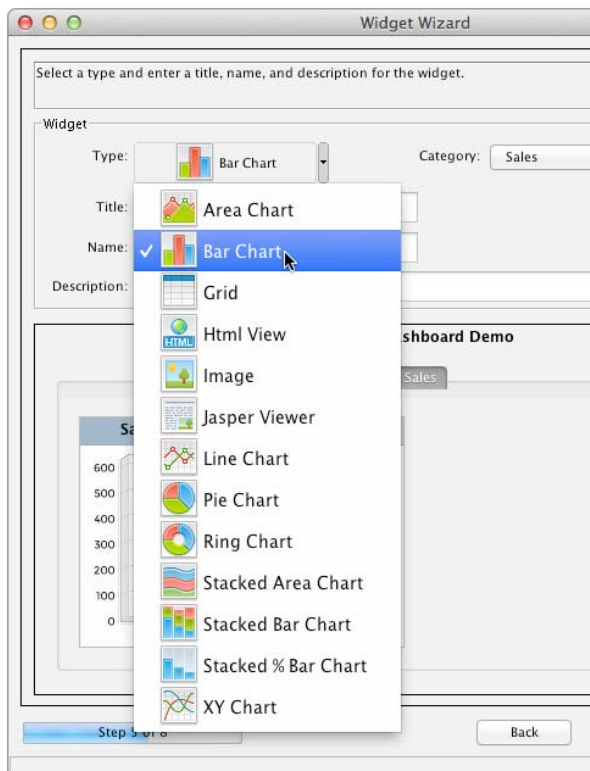


Figure 4. The available widget types

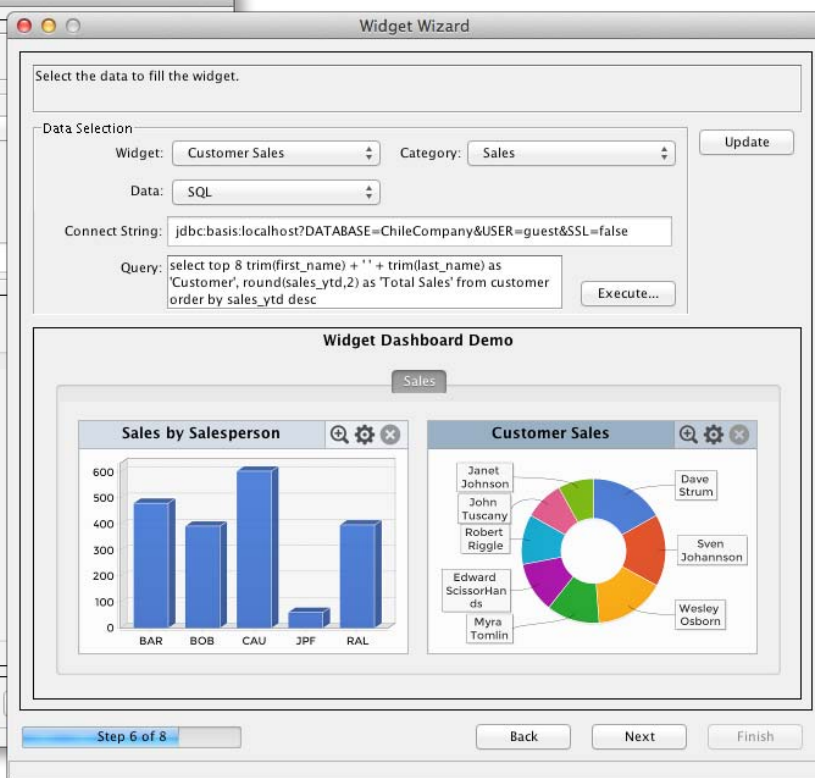


Figure 5. Select the data to fill the widget

from a basic chart such as area, bar, line, pie, or ring charts, to more advanced chart types such as stacked area, stacked bar, stacked percentage bar, or XY charts. The Widget Wizard also includes a 'Jasper Viewer' widget for displaying JasperReports, an 'Image' widget for images, and an 'Html View' widget for displaying HTML. You can even put a 'Grid' in your widget. Icons in the 'Widget Type' control provide a visual representation of each widget type. See Figure 4.

Obtaining Data

The most time consuming part of widget creation is figuring out what data is needed for the selected widget type and how to get it. Different widgets use different methods for obtaining the data. An 'Image' widget uses either a path or URL to the image, an 'HTML View' widget uses HTML text or a URL, and a 'Jasper Viewer' widget uses a connect string and JasperReport file for data. Data for the various chart widgets can be obtained either by SQL or a recordset. Depending on your use case, you may choose not to define data for the widget, in which case you can skip this step and proceed to the next wizard screen. For example, you may wish to integrate a widget into an existing application by taking advantage of the code that the Widget Wizard generates. In this scenario, the existing application already has the requisite data, so the Widget Wizard does not need to obtain the data again when creating the widget definition.

Figure 5 shows the 'Connect String' and the 'Query' string required to use SQL. The 'Connect String' defaults to the **ChileCompany** database on the local machine as the guest user. You can easily modify this to connect to your database and machine with the appropriate credentials. To view query results, select [Execute] after entering a query.

Once your query gives you satisfactory results, select the [Update] button to update the widget and fill it with data. If there is an SQL issue, a message box appears with the type of results required for the particular type of widget, and with a tip for the SQL query. See Figure 6.

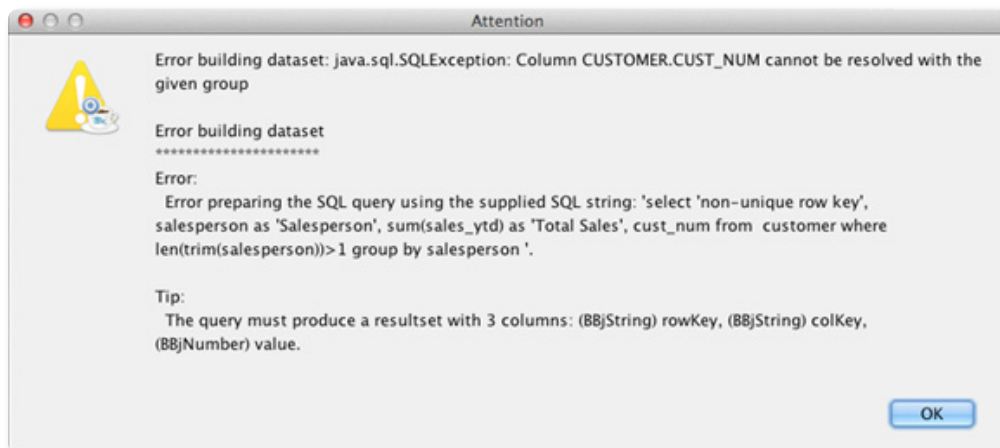


Figure 6. A message box showing the error and a tip

Code Generation and Execution

The end of the Widget Wizard bears the fruit of this process. You are ready to generate the BBJ OO code and let the wizard write it out to a source file that you specify. The Javadoc-commented source code includes an error handler and is ready to launch if you select either of the 'Run in GUI' or 'Run in BUI' checkboxes. See **Figure 7**.

Clicking the [Next] button causes the wizard to actually generate and save the OO code. The generated code contains a public class that creates the widgets, which a BBJ program can then access to instantiate and display the widgets. The code also includes a short sample at the top of the file (shown in **Figure 8**) that does exactly that so that your widgets actually run.

The public class definition, partly shown in **Figure 9**, contains all of the code necessary to replicate all of the widgets and categories you defined in the previous wizard steps. The code is complete as-is, but can be easily augmented to further modify any of the widgets defined within. For example, you can find your chart-based widget object in the program and add a couple of lines of code to modify the chart's colors or set a background image. Download the code sample at links.basis.com/14code.

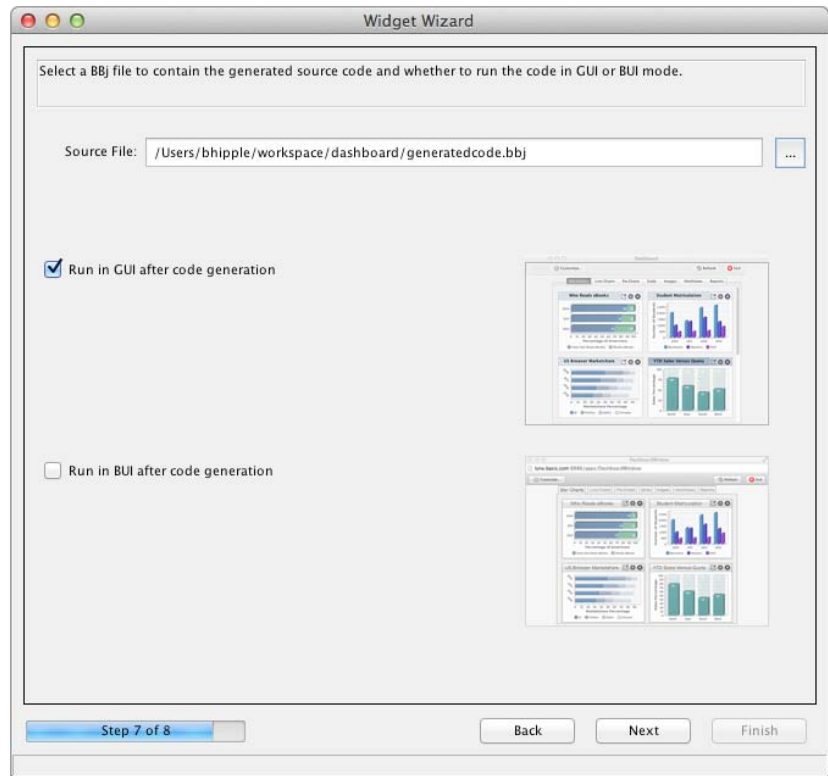


Figure 7. Specify a file for the generated code and choose how to run

```
rem /**
rem * Sample code
rem */
BBJAPI().getConfig().setOptionSetting("ERROR_UNWINDS",1)
seterr ErrorHandler
declare WidgetDashboardDemoWindow widgetDashboardDemoWindow!
rem Embedded Code Begin
use ::/Users/ndecker/Desktop/WWTest1.bbj::WidgetDashboardDemoWindow
widgetDashboardDemoWindow! = new WidgetDashboardDemoWindow()
widgetDashboardDemoWindow!.doModal()
rem Embedded Code End
release

ErrorHandler:
x=MsgBox(errmes(-1))
release
```

Figure 8. An excerpt from the generated code that instantiates and displays the widgets

```
rem /** WidgetDashboardDemoWindow
rem * Generated class
rem */
class public WidgetDashboardDemoWindow

    field public Dashboard Dashboard!
    field public DashboardWindow DashboardWindow!
    field public DashboardCategory SalesDashboardCategory!
    field public DashboardWidget SalesBySalespersonDashboardWidget!
    field public PieChartWidget SalesBySalespersonWidget!

    rem /**
    rem * Constructor WidgetDashboardDemoWindow
    rem */
    method public WidgetDashboardDemoWindow()
        rem Create the Dashboard
        dashboardName$ = "WidgetDashboardDemo"
        dashboardTitle$ = "Widget Dashboard Demo"
        #Dashboard! = new Dashboard(dashboardName$,dashboardTitle$)

        rem Create DashboardCategory
        categoryName$ = "Sales"
        categoryTitle$ = "Sales"
        #SalesDashboardCategory! = #Dashboard!.addDashboardCategory(categoryName$,categoryTitle$)
```

Figure 9. The class definition in the generated code

The summary screen displays the code necessary for embedding the widget inside your application logic (see **Figure 10**). You can easily copy this code from the edit box and paste it into your program.

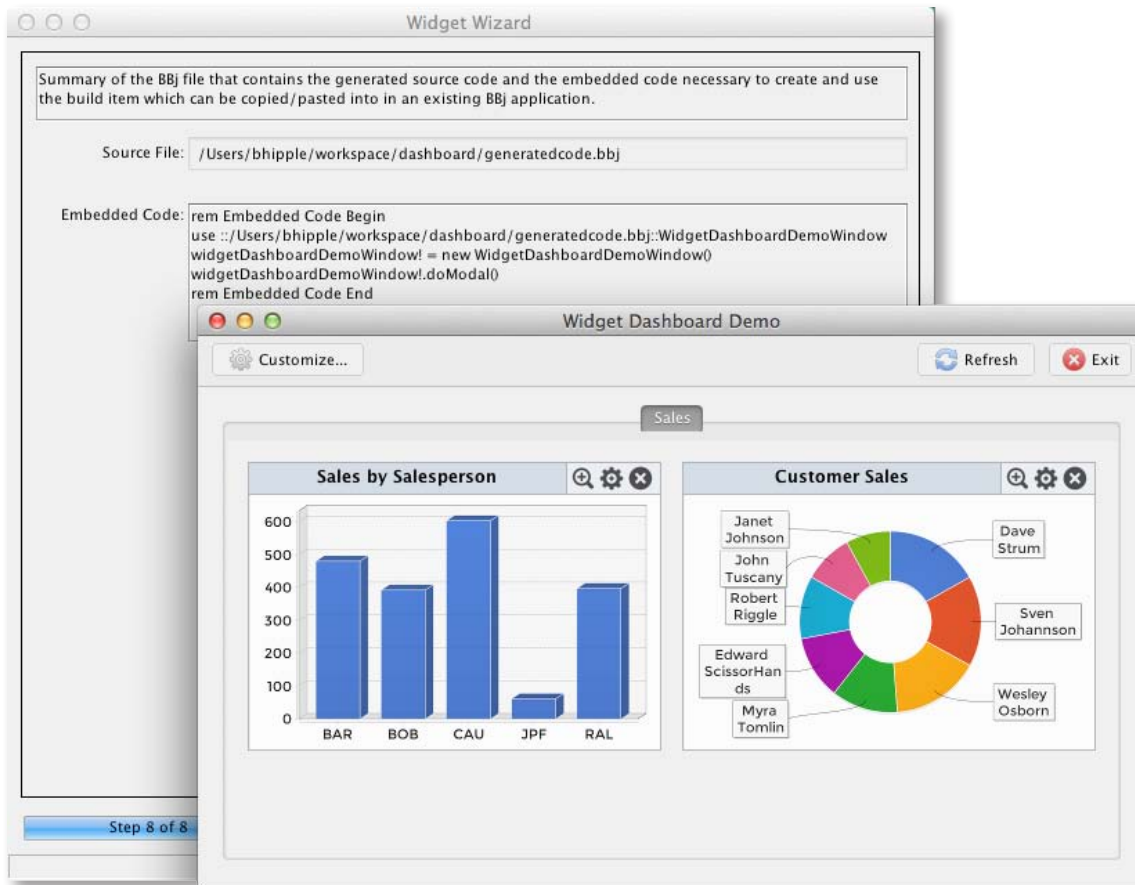


Figure 10. Running your code in GUI with a summary display

Summary

Using the Widget Wizard as a sandbox to create widget code is a fantastic way for you to get familiar with using widgets. To try it out, go to links.basis.com/getbbj and select the newest product for download. For assistance running the wizard, check out the tutorial-in-process at links.basis.com/widgetwizard. It has been our experience that once you create the first couple of widgets, creating more becomes a snap. The Widget Wizard eliminates the learning curve – like magic! ■



- Read related articles in this issue
 - [Easier Decision Making With the Dashboard Utility](#)
 - [Dash Boredom With the Dashboard Utility](#)
 - [AddonSoftware's Digital Dashboard Takes Off](#)
- Refer to [Dashboard Utility Overview](#) in the online Help
- Watch these past Java Breaks
 - [Adding the New Digital Dashboard to Your App](#)
 - [Embedding Widgets in Your BBx App](#)
 - [The New Widget Wizard - Dashboards and Widgets Without Any Code!](#)
- Download the [code sample](#)



Car/IT Integrates a 'New Model' With BBj

For nearly 15 years, Audev's Car/IT has been the software application of choice for many car dealerships throughout the Netherlands, Germany, Bulgaria, and Mexico. Its power and popularity primarily lie in its two main modules named as the Dealer Management System (DMS) that manages all administrative tasks in the dealerships, and the Customer Relationship Management (CRM) module, which addresses all marketing-related activities.

The origins of these modules date back nearly 30 years. In the late 1990s, Audev rewrote their DMS, originally based on an application from the 1980s, with Visual PRO/5® and PRO/5 Data Server®. In early 2000, Audev integrated the Visual Basic 6-developed CRM with the DMS. Needless to say, there was a great opportunity for enhancing these modules and addressing the disadvantage of maintaining two databases in one application and developing in two different languages. With BASIS technology moving forward so fast and offering such a variety of new and exciting features, Audev took the step to fully upgrade Car/IT to the BBj® environment and eliminate the Microsoft technology components.

Read on for a close look at how Audev, in their own words, modernized their application with BBj.

Two years ago, we took our first step with the release of the DMS Workshop Planning program in BBj. By staggering the transition process, we were able to ensure our programs were compatible with the new environment, while at the same time learning about what opportunities BBj offered us.

During the transition, the excellent team at BASIS, from engineers to management, supported us by helping us ensure that our product stays ahead of the competition and fulfills the specific needs of our customers. While a complete transition is challenging, it also offers a unique chance to look at our concept afresh, and redefine aspects of the application.

Business Partner

A long-held goal of ours, for example, was to ensure that the Business Partner of a car dealership was always at the heart of the application. And that is what we achieved with the Business Partner program, which manages all of the information that is available to help dealers to provide efficient and professional service to their customers. The program displays a 'tree view' on the left with relevant information such as cars, contact plans, and order history (see **Figure 1**).

To manage the Tree control, we designed and used a custom BBj class object. This hides the details of the original component and provides an easy interface to implement in other parts of the application. Depending on the role of the users, they can access



Wimco Driesse
CIO Audev BV

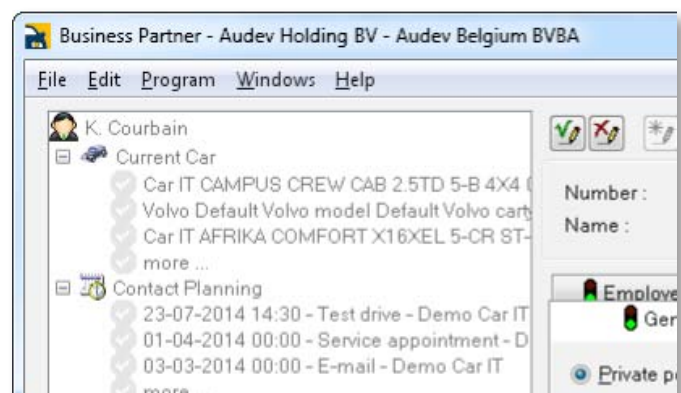


Figure 1. The business partner as the central object

key functions from the tree with a right mouse click as shown in **Figure 2**.

Across the top of the main screen are a number of tabs with related information, all available to users according to their access rights. On the 'General' tab, we store information about social media accounts to which it will be possible to send messages via the Car/IT Communication Handler (see **Figure 3**).

Diary

The second major redesign was the 'Diary' program (a calendar program with scheduling support) as shown in **Figure 4**. BBJ made it easy to develop this program that enables users to create and manage meetings, appointments with their business partners, and other events.

Google Apps Integration

We are working to implement a calendar that more closely matches the look and feel of Google Calendar. To accomplish this, we are collaborating with the BASIS engineers to update the BASIS-provided GApps building block utility to the new Google Calendar API version 3.0. This allows us to reach our goal without having to develop that functionality ourselves.

The integration will be a two-way process between Google's calendar and the Car/IT calendar. For example, if Karl makes an appointment in the Car/IT calendar, the program updates his Google calendar, and when he creates an appointment in his 'external' calendar, it appears in the tree view as an imported appointment. From here, he can invite a business partner to the appointment and from then on, the calendars are linked and updated in both directions.

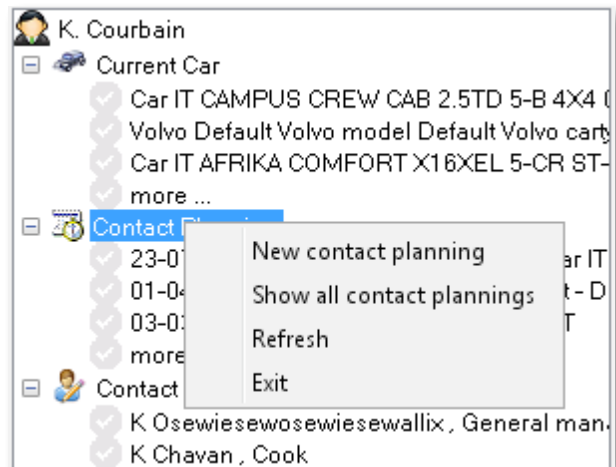


Figure 2. In the tree, a right mouse click brings up a pop-up menu

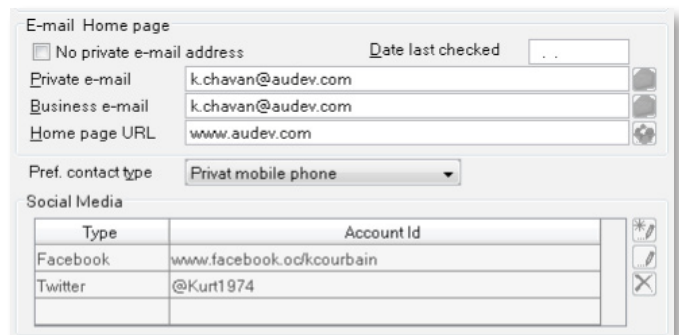


Figure 3. The General tab contains communications options including social media accounts

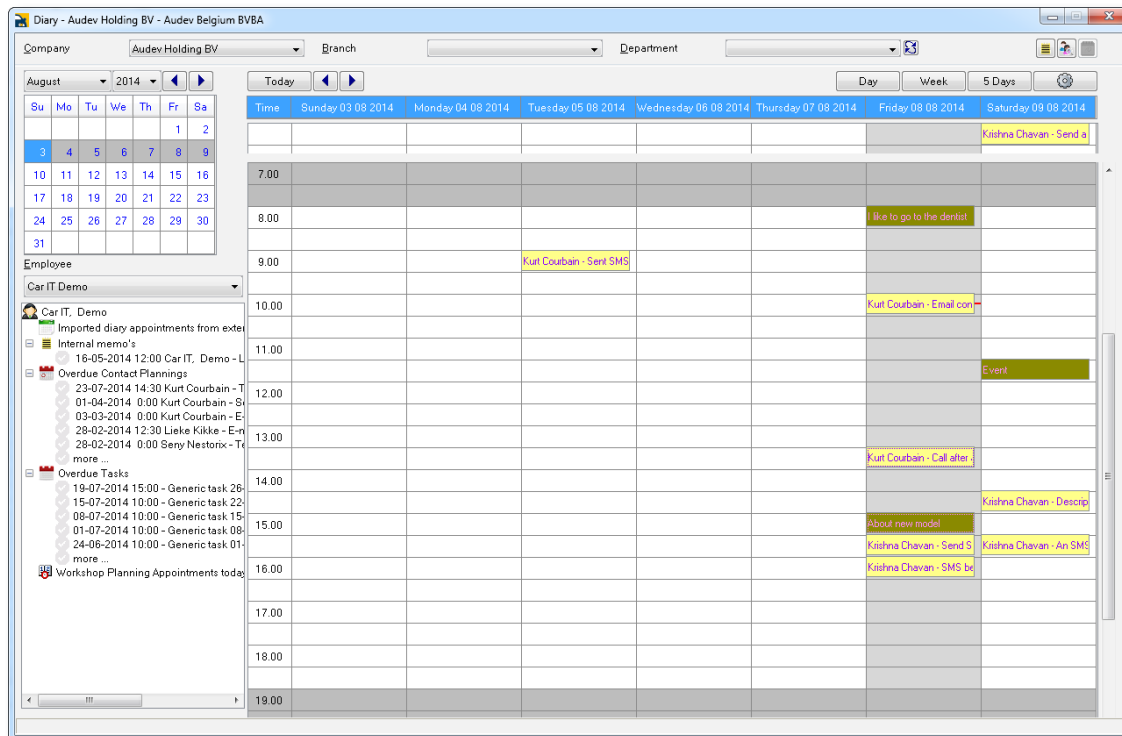


Figure 4. Car/IT Diary program

Layer Functions

In addition, we created a component for the 'Diary' program that handles all the user interface layer functions. A user can easily drag and drop diary events within Timed and Not Timed zones. Events scheduled for the same time will appear split over the available display area. To implement this component, we created multiple BBJ custom classes and are also using multiple Java classes to maintain the data in memory. This gives optimal user performance even in a multi-user system with minimal hardware capacity.

Grid Enhancements

One critical component of the 'Diary' program is the grid. BASIS engineers enhanced the BBJGrid control with new functions so we were able to implement all our requirements. We developed all of the other functionality in-house without any notable problems. When issues arose, BASIS engineers were able to help us solve them within BBJ.

Menu

In examining the existing DMS, we decided that the menu looked outdated; **Figure 5** shows the legacy menu.

To give it an improved look and feel, we replaced it with a new, modern multiple document interface (MDI) menu system. In an MDI menu, tasks launched from the menu's MDI parent are separate invocations of BBJ but they live inside the menu's workspace. This ensures that the processes are tied together and show as a single task on the Windows desktop. This MDI menu system takes into account the existing rules for security while still maintaining the company concept of Car/IT.

Here again, the BASIS engineers provided assistance that was very helpful. We adopted some existing components from the [BASIS GitHub](#) repository and moved forward quickly. This part of the project is still under development, but **Figure 6** gives you a sample of the 'new' menu panel of the MDI.

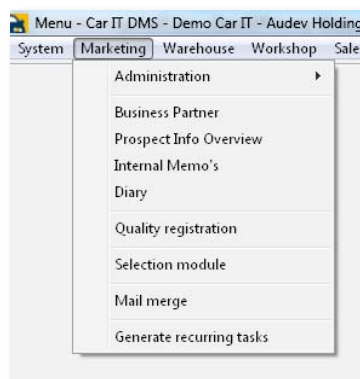


Figure 5. Legacy Car/IT menu

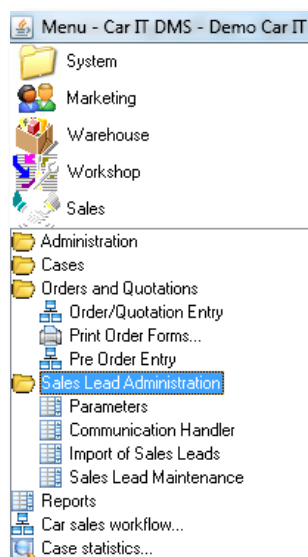


Figure 6. Prototype of the new MDI Car/IT menu panel

Summary

In the next few months, we will release our integrated DMS/CRM version of Car/IT. Combining a new look and feel, a great team of Audev engineers, and the support of BASIS engineers – and based on BASIS' BBJ technology – Car/IT is ready for the future.

But ... we're not done yet! We already have other ideas in mind like extending parts of the application to mobile devices, moving to a browser user interface, and many more. With so many exciting opportunities offered by BBJ technology, the road ahead for Car/IT is a long and interesting one! ■

Audev, founded in 1989, develops software for the automotive industry using the talent and experience of developers with roots in both automation and the automotive sector. Audev's software, Car/IT, is the first open DMS capable of handling multiple makes, multiple manufacturers, multiple suppliers and multiple languages. Its leading edge technologies allow Car/IT to fully integrate and interface with the communications tools and applications programs of the various manufacturers. Long-lasting relationships with the users of Car/IT products, established by leasing and not selling the software, ensuring that Car/IT suits the needs of every dealership, for every dealer size, for every brand, in every language. www.audev.com



Wimco Driesse, CIO of Audev since 1999, has been active in the automotive industry for more than 18 years. Wimco is responsible for all new product development in Europe, Latin America, and the US.





*By Nick Decker
Engineering
Supervisor*

BDT Tips for Less Pain and More Gain

The CodeEditor in the BDT (Business BASIC Development Tools) Eclipse plug-in is a powerful, full-featured editor for BBJ® programs. What makes the BDT one of the best options for developing BBJ programs is that it not only inherits a number of features from the DLTk (Dynamic Languages Toolkit) framework on which it is built, but BASIS engineers enhanced it further by adding several supplementary capabilities specific to the BBJ language. As with most sufficiently advanced editors, it's capable of so many functions that often times new users are not aware of all that is available. Worse yet, Business BASIC developers may not be using the CodeEditor to its fullest, and may be missing out on capabilities, time saving tips, and other productivity enhancements that 'power users' put into practice on a daily basis. This article does not attempt to cover the myriad of options that the BDT offers, but rather covers a few of the 'can't live without' features that we have compiled after polling several BASIS engineers who spend a good chunk of their day inside the Eclipse IDE.

Quick Access to Anything

Eclipse can do innumerable things without a mouse like using only keystrokes to access menu options, tool buttons, and more. Many options and commands are available that, in all likelihood, most programmers don't know about or wouldn't know how to get to. For example, press [Ctrl]+[Shift]+L, or [Cmd]+[Shift]+L (⌘⇧L) on OS X, to see the myriad of key commands that are not only available, but customizable to your liking. Finding the right keystroke or command may seem like looking for a needle in a haystack, but Eclipse offers a great way to give you easy access to whatever you're looking for with the Quick Access search option. Located at the top right of the toolbar, you can type in **bb** as shown in **Figure 1** to display a list of commands, menu items, preferences, and more that contain the letters **bb**.

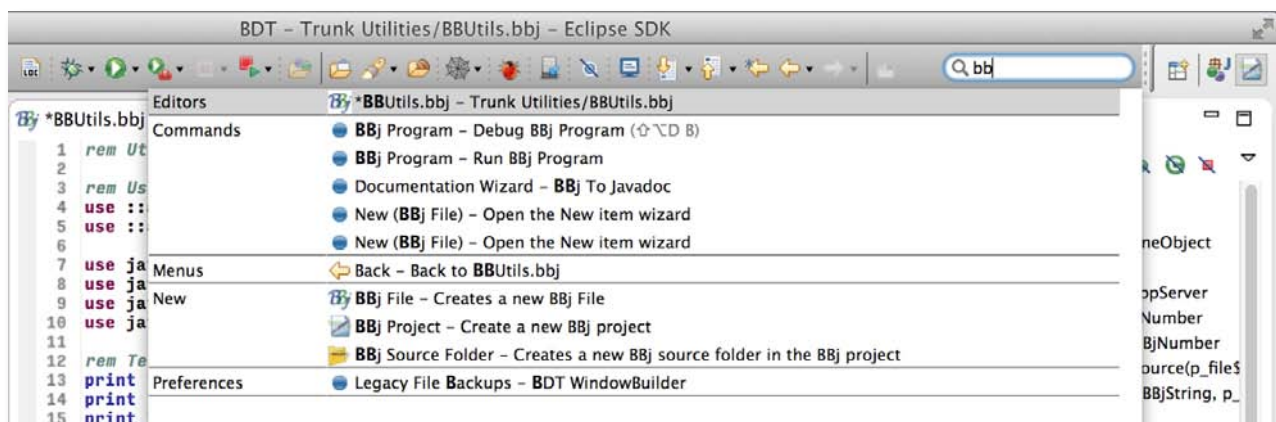


Figure 1. Fast access to almost everything is available from the Quick Access search

In practice, this Quick Access search feature is a huge timesaver. For example, you can type the word **font** into the search bar (shown in **Figure 2**), then select the first 'Commands' option to display the 'Fonts and Colors' preferences dialog. You can access the desired preference pane in just a few keystrokes using the Quick Access search, which is much faster than using the mouse to navigate the menu and preferences hierarchy, even assuming you know where the command resides in the first place.

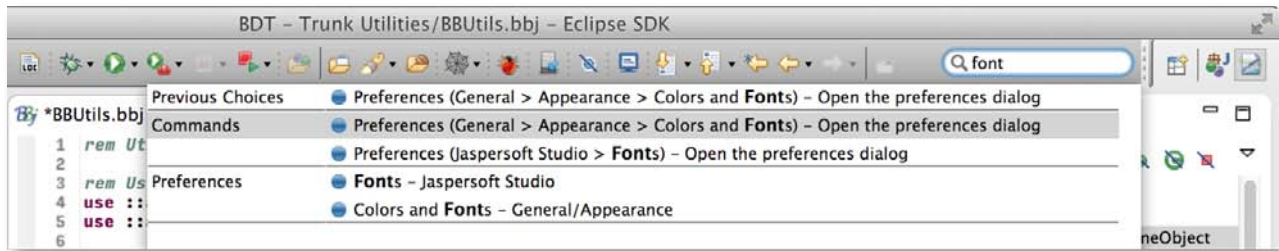


Figure 2. Quick access to the Colors and Fonts preference is just a few keystrokes away

Program Navigation – Outline View

Eclipse's 'Outline' view is at the top of most programmers' list of favorites because it provides an efficient way to navigate through a program. In a standard BBJ program, it lists variable declarations, program labels, and functions. In an object-oriented program, it's even more useful as you can drill down into classes to see methods as well as class and method field variables. Everything has a unique icon shape and color so you can readily tell at a glance the difference between classes and interfaces, as well as private, protected, and public methods and variables.

The tree node entries are chock full of information too. For example, instead of just listing all possible methods in a class, the entries display each method's parameter list of variables and their type, and the return type of the method. **Figure 3** demonstrates this with the `applyCss()` method that appears highlighted in the outline view and selected in the CodeEditor.

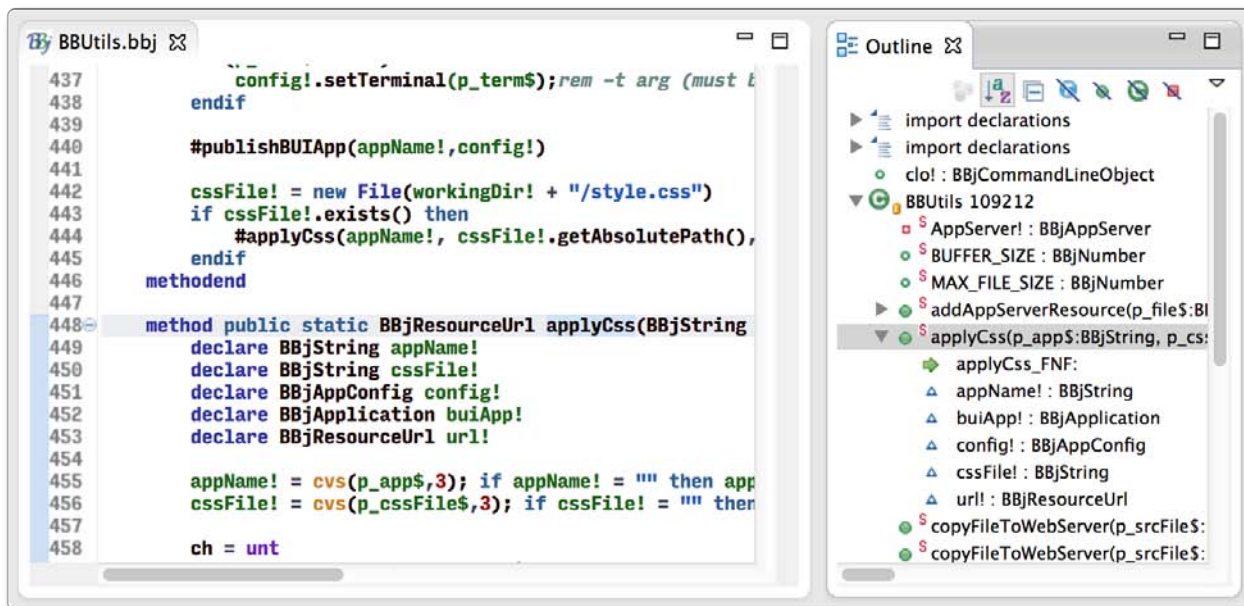


Figure 3. Navigating to a method in the CodeEditor (left) using the Outline view (right)

The Outline view's toolbar icons located at the top of the view make it even more flexible, allowing you to sort the view contents alphabetically, hide/show components, expand/collapse nodes, and more.

Program Navigation – Bookmarks

Bookmarks are another great way to quickly navigate through code and find often-used methods or routines. Simply select one or more lines of pertinent code, then select the [Add Bookmark...] option from the 'Edit' menu, and enter a meaningful name in the prompt. All of your saved bookmarks will appear in the 'Bookmarks' view as shown in **Figure 4**. If you do not see the bookmark view by default, go to the menu and click on Window > Show View > Other > Bookmarks, then place the view anywhere you desire in the IDE for quick reference.

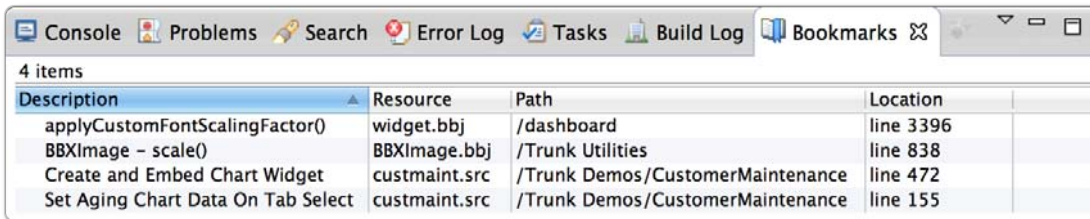


Figure 4. The bookmarks view showing saved bookmarks from multiple programs

The bookmarks are associated with the program and selected code, so opening up a bookmark from the list causes the BDT to load the appropriate program, jump to the code of interest, and even select the original lines of code. You can freely add or remove code in the original program and the bookmark will still be valid, as Eclipse updates its location dynamically to stay in sync with any insertions or deletions.

Editing – Multiline Comments

The BBj language uses the **REM Verb** to indicate comments in a program, but what about the scenario when you would like to comment out a large block of code? Programmers often write test code intended to replace sections of their original program, but using the REM Verb to remark individual lines can be burdensome. The CodeEditor has an elegant solution to this, as you can highlight any number of lines of code, then select Source > Toggle Comment or the appropriate hotkey: [Ctrl]+/ in Windows and Linux or [Cmd]+/ (⌘/) on OS X. This instantly adds or removes REMs to every selected line, making it quick and easy to remark multiple lines of code at once. **Figure 5** shows the same block of code before and after toggling comments in the editor.

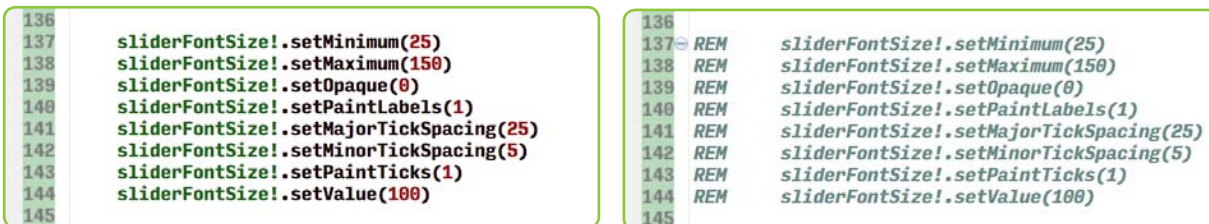


Figure 5. A block of code before (left) and after (right) toggling comments

Editing – Multiline Editing

Block selection mode is another sometimes little known favorite that can really be a time saver. You can simply toggle block selection mode using the keystrokes [Alt]+[Shift]+A on Windows and Linux or [Cmd]+[Opt]+A (⌘+A) on OS X. Turning on this editing mode allows you to select rectangular blocks of text rather than wrapped lines to easily delete, move, copy, or paste these blocks just as you would any other selection. After turning on block mode the font may change, depending on your configuration for the 'Text Editor Block Selection Font', indicating the new selection mode. You can then make rectangular selections (see **Figure 6**) where the selection is a one-character column from lines 412 to 419.



Figure 6. Selecting a column of text in block mode

After selecting the column, you can type in text that is then inserted on all eight of the selected lines simultaneously. **Figure 7** shows how we inserted the 'sql\$ = sql\$ + ' text on all of the lines at once, completing the code and resolving the syntax errors indicated in **Figure 6**.

```

411 sql$ = "SELECT cast(invoice_date as SQL_CHAR) as invdate, sum(total_sales) as TOTAL_SALES "
412 sql$ = sql$ + 'FROM ART03 '
413 sql$ = sql$ + 'WHERE '
414 sql$ = sql$ + 'firm_id = '01' '
415 sql$ = sql$ + 'and '
416 sql$ = sql$ + '(invoice_date between ' + inputDStart!.getText() + ' AND ' + inputDFinish!.getT
417 sql$ = sql$ + 'and '
418 sql$ = sql$ + 'slspn_code = ' + salesperson!.get(salesperson$) + ' '
419 sql$ = sql$ + 'GROUP BY invoice_date '
420

```

Figure 7. Inserting text into eight lines simultaneously

Editing – Templates Do Your Typing

Developers usually find themselves typing certain code patterns frequently, such as the familiar FOR/NEXT verb loop or IF/THEN/ELSE conditional statements. Templates in Eclipse are similar to word processor macros, as they transform a few characters into predefined blocks of code. This not only saves time, but the resultant code is already indented and formatted, thus improving consistency. The BDT comes with several default templates that you can access via the Content Assist keystroke combination of [Ctrl]+[Space]. For example, type `if` and invoke Content Assist, after which you can select the desired built-in template. **Figure 8** shows what this looks like in the editor.

Highlighting a template in the popup window results in a preview of the defined code block, and selecting the template inserts the code into our program. Templates also save time by eliminating our need to position the cursor with the mouse. In the example above, after inserting the code into our program, the template selects the **condition** text. That means we can type in the appropriate condition for our program, which overwrites the placeholder text. Additionally, pressing the [Return] key causes the template to advance the cursor intelligently in between the IF/ENDIF lines. Not only did the template leave the THEN portion intact, it also correctly indented the cursor.

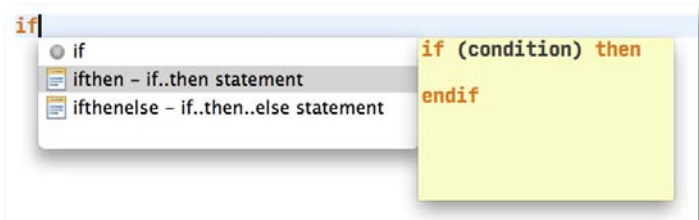


Figure 8. Using a template to insert a block of code

To get the most out of templates, create your own to help automate your development process. Templates can take advantage of pre-defined variables such as `${cursor}`, which specifies the cursor position when exiting edit mode, or `${word_selection}` that resolves to the content of the currently selected text. These variables allow you to add a level of sophistication and flexibility to your templates so that they will be more valuable than simply pasting in static blocks of code.

Editing – Revision Comparisons

On occasion, every programmer has broken a piece of code that used to work perfectly. Usually this occurs after hours of work and many changes to the code, making it a challenge to know which change is responsible for the problem. If you're using a source code repository, you can use Eclipse to compare your current code to the latest version in the repository. But what about when you're working on code that you haven't checked in yet? Eclipse handles that case too, as you can right-click on the program, select Compare With > Local History to open the 'History' view. Eclipse keeps a running list of revisions that are timestamped and available in the History view so that you can see a list of your recent changes and compare the current code to one of those versions. You can even configure the Local History to set limits such as retention time, saved changes per file, and maximum file size in your workspace preferences.

Editing – Quick Diffs

Side-by-side comparisons are definitely useful, but if you want a quicker, easier way to see the changes you've made in your code, use the real-time Quick Diff. You can configure Quick Diff in preferences and set the various colors for changes, additions, and deletions as well as configuring the reference source. You can set the reference source to 'Version on Disk', but comparing local copies to the latest in a source code repository yields much more functionality. After configuring it to show in the ruler, both the left and right edges of the CodeEditor will display colored blocks, visually depicting changes to the

code. In **Figure 9**, red corresponds to code deletions, green to additions, and blue to changes in the existing code. The left ruler shows changes for each adjacent line and the right ruler shows changes for the program as a whole. More specifically,

- **Red number 1** indicates a place where code has been deleted, and after hovering your mouse over that block, a popup appears showing what the deleted code block looked like. If you want to undelete it, just copy it from the popup and paste it back into the program, or right click and select [Restore Deleted Lines].
- **Green number 2** shows that this section of the code is new compared to what is checked in to the repository.
- **Blue number 3** shows other places in the program where code has been changed.

Hovering over a block on the right ruler in **Figure 9** shows information about the section, and notes in a popup in the lower right that there is a new line label and it added 22 lines. Clicking on any block in the right hand ruler jumps immediately to that section of the program and selects the new or modified code. In this way, the Quick Diff not only shows you changes to your code at a glance, but it also serves as a rapid way to navigate in your program and jump to other modifications in the file.

Memory Management

Eclipse is a multi-purpose development environment and you may have dozens of different modules besides the BDT installed to work

on multiple projects, each with different editors and plug-ins. Memory usage can have a direct impact on performance and Eclipse gives you a way to keep an eye on memory consumption. You can mark the General > 'Show heap status' checkbox option in Eclipse's preferences to display information about the current Java heap usage. The heap status is located in Eclipse's bottom status bar, as shown in **Figure 10**.

The graph and numbers indicate how much memory Eclipse is currently using versus how much memory you have allocated to it. It also includes a garbage can icon that when clicked performs a garbage collection to free up any reclaimable memory.

If you find Eclipse constantly running close to its maximum heap allowance, you can increase its heap size as detailed in the FAQ [How do I increase the heap size available to Eclipse?](#) Depending on your usage of the IDE and your machine's amount of physical RAM, increasing the amount of memory available to Eclipse can dramatically improve performance.

Summary

Eclipse is such a powerful and capable development environment that detailing every aspect and feature would take an entire book (of which there are plenty, not coincidentally). This article took a different approach, and limited its scope to a select set of features that BASIS engineers use on a daily basis. If you have not heard of some of them, or have yet to put them into your development cycle, give them a try today and see how they can increase your productivity. ■



- Read these *BASIS International Advantage* articles:
 - [Have it Your Way With New BDT Preferences](#)
 - [Eclipse: The Toolset of the Future](#)
 - [Double Your Pleasure With Eclipse Plug-in Documentation](#)
- Check out these additional resources:
 - [BASIS documentation](#)
 - [Eclipse documentation](#)
 - [FAQ How do I increase the heap size available to Eclipse?](#)

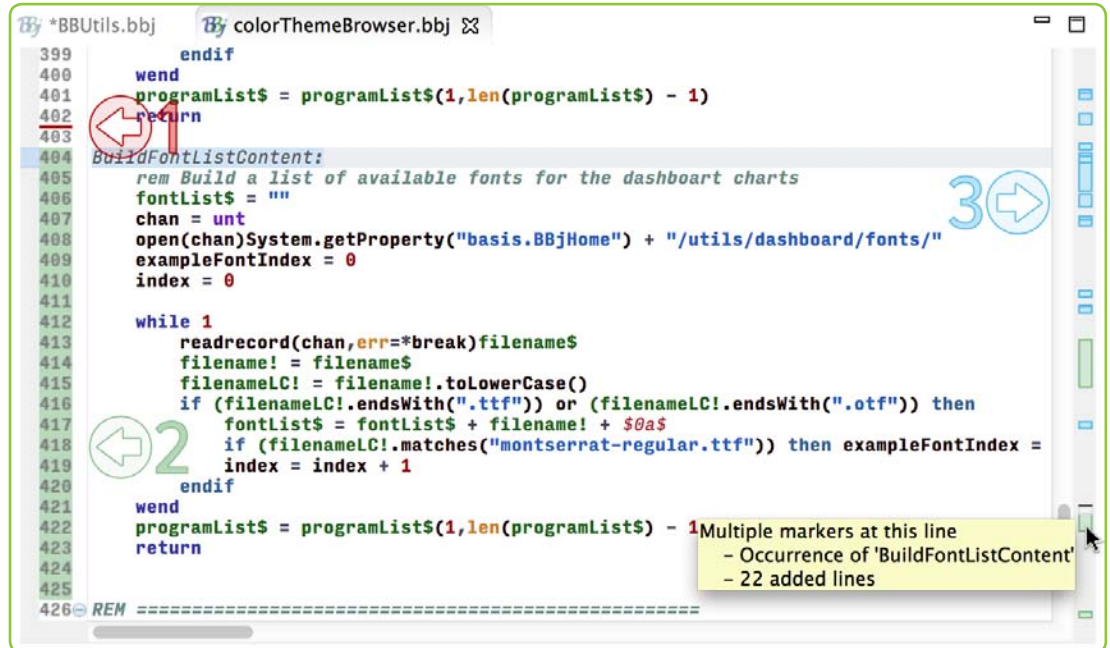


Figure 9. The Quick Diff ruler entries showing changes to the program

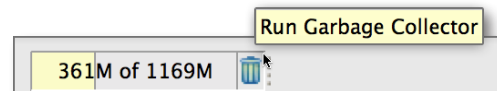


Figure 10. The amount of currently used versus allocated memory in Eclipse



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.

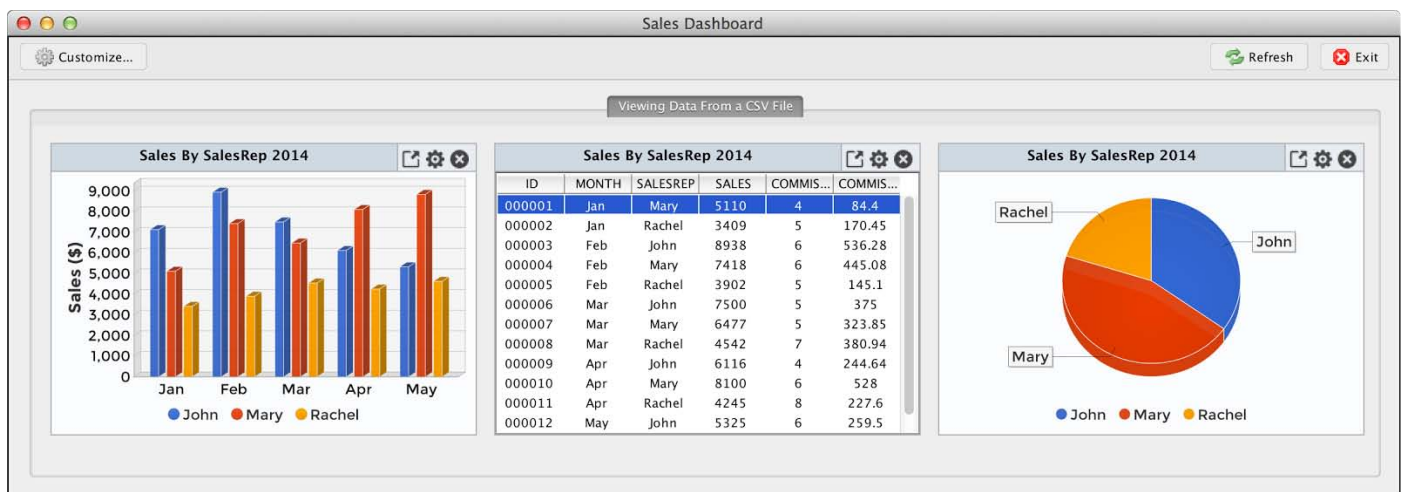


Figure 1. The dashboard program built during the Java Break

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**.

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



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.

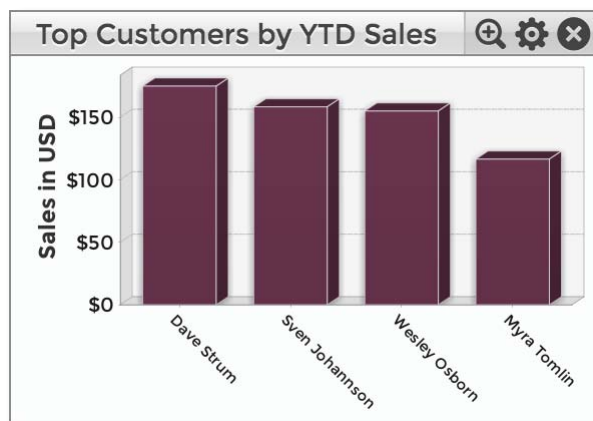


Figure 4. A dashboard widget that was filled automatically given an SQL connection and query

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.

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

CustNum	Name	Rep	Total Sales
000006	Dave Strum	CAU	165.20
000060	Sven Johansson	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

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](#)

BASIS International
Advantage
 Missed an Issue?
www.basis.com/advantage



Putting Your Software Through its Paces

Like all software products, AddonSoftware® by Barista® is an evolving complex system with many parts that make up the whole. The AddonSoftware business logic is the brains that control what it does, and the BBj®/Barista framework is how it gets that job done. With every change to either part of the system, the possibility of an error or incorrect logic entering into the system becomes greater. A robust testing procedure that uses tools that are consistent and logical finds such defects before they become big problems.

In traditional software testing, the developer performs the Component and Unit testing stages. Then the Quality Assurance team does System Integration testing, which can use up a significant amount of people, time, and money executing manual tests. Using an automated testing system is a better use of resources, exercises the software consistently, and finds defects more quickly. In fact, when an abnormal action or error happens, it can even automatically notify the testing team of a problem!

BASIS recognized the value of such a testing procedure and selected Quality First Software's QF-Test (www.qfs.de) for automated integration testing. QF-Test is a Java-based professional tool to automate testing of Java and Web applications with a graphical user interface, and it helps BASIS to find bugs faster.

Testing Infrastructure

A consistent software testing methodology in a modern programming environment provides critical feedback to both the developers and management that a quality product is being produced for the public. In a complex system like AddonSoftware by Barista, the chance of abnormalities showing up in even a well-designed, logically thought out design become even greater. Large or small, these defects need to be caught early in the development cycle to allow the developers time to correct them and to reduce the repair costs.

Using an automated testing framework such as QF-Test results in faster delivery of a cleaner product, a win-win for all. Customers receive a better product and BASIS can better use resources to address issues that do make it into the field.

Of course, there is nothing like the real world data and procedures to expose issues in the software that the engineers did not design for or anticipate. That is why BASIS includes sample databases in the downloadable BASIS Product Suite .jar file. Testing with the same set of consistent data in the databases allows BASIS to create and execute tests that can depend on predictable data and software responses. This allows QF-Test to execute tests on Barista and AddonSoftware designed to exercise as many of the key functions as possible with predictable results. When the results don't match what is expected, the test is flagged as having an abnormality to indicate that it needs to be examined further. Not all abnormalities are problems, some are minor but could be indicative of larger problems. In software testing, having as many eyes as possible on the product is a good thing. With the use of automated testing the process of finding problems before they are issues is much faster, more thorough, and repeatable. The result is a much more solid final product release.



Brian Sherman
Software Developer

Executing Test Suites

QF-Test controls the system under test by using test scripts called *Test Suites* to drive the testing sequences. A Test Suite is a collection of action steps that make up a logical sequence. Breaking up the steps into a series of *Test Case* steps makes it easier to debug the testing script when an unusual action occurs during execution.

Each Test Suite consists of a series of actions: Setup (start), the Test case execution, and Cleanup (stop). Each of these actions performs a critical role in the testing process. Setup is where QF-Test creates the test environment. Here, QF-Test sets the global variables, starts Barista, logs in, and takes control. Next, it starts to process the action steps of the tests. As it steps through individual tests, QF-Test checks for multiple conditions such as whether the forms and data display correctly, whether the prompts appear in the correct order, and if they appear within a reasonable time. If any of these conditions don't occur as expected, QF-Test stops execution and displays an error. If the Test Suite completes without any errors flagged, QF-Test executes the Cleanup section and shuts down the system under test.

The example shown in **Figure 1** runs through the AP to GL cycle. This Test Suite tests such key Barista components as Header/Detail Entry Grids, Document Processing (PDF and Jasper Reports), and of course the AddonSoftware business logic. It also tests for “under the hood” components such as displaying information, timing of actions, and operating system interactions such as cursor movement and keyboard function keys. As you can see, it logically steps through the business logic and performs the entry of an AP Invoice, selection and payment. With printing the ‘Registers and Updates,’ it verifies that the data flows through the AP module to the GL module.

Reporting Test Results

When QF-Test encounters an abnormality during execution of a Test Suite, it stops and brings up the display as shown in **Figure 2**. This shows a ‘Component not found’ type of error message window. In addition, QF-Test can automatically start its internal debugger sub-system to assist with tracking down the problem. This error occurred when the QF-Test was unable to properly start and connect to the AddonSoftware client.

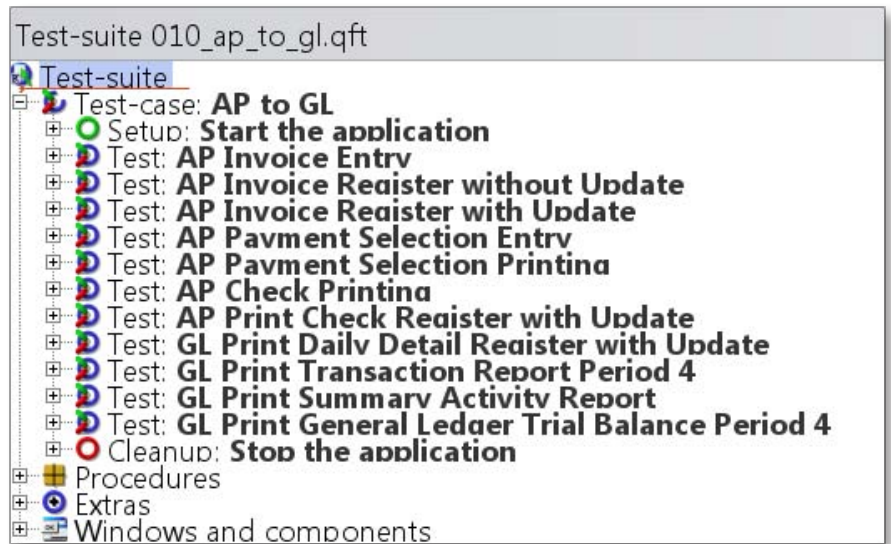


Figure 1. Test suites for AP to GL

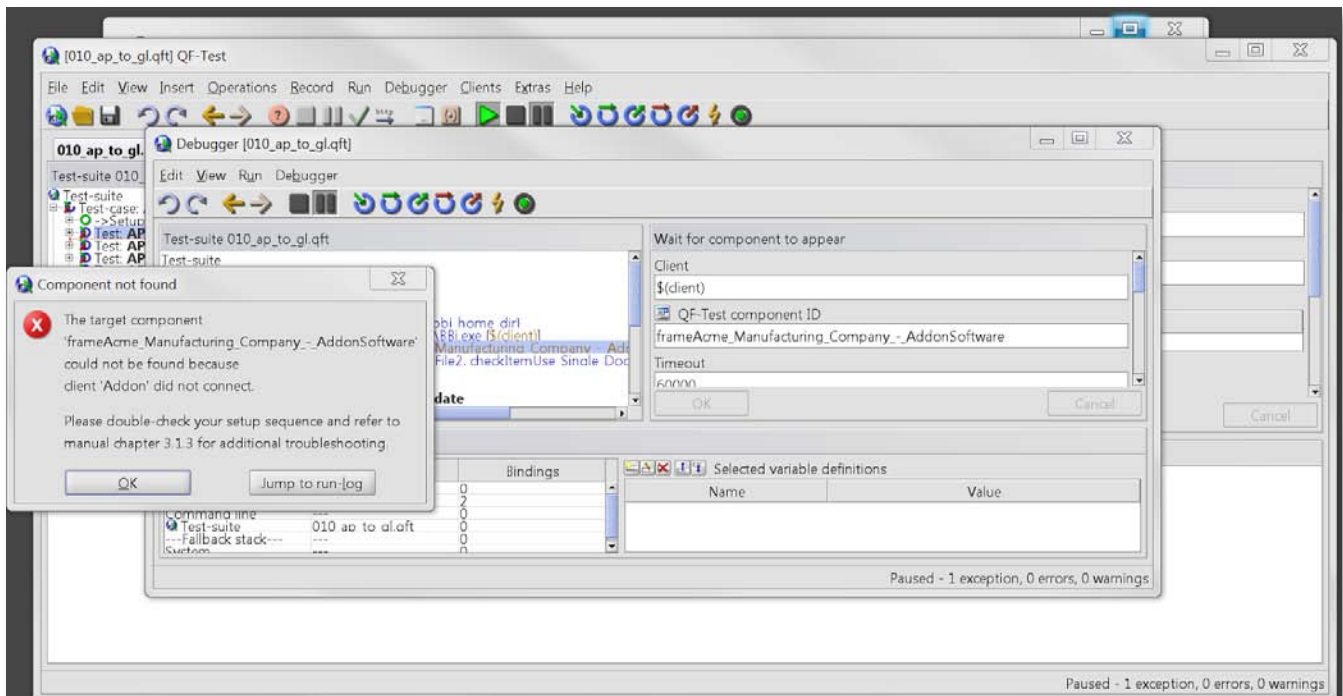


Figure 2. QF-Test 'Component not found' error

In **Figure 3**, QF-test has completed the Test Suite and reported '0 exceptions and 0 errors' as shown in the lower right corner of the Test Suite results. The number of warnings shown indicates that there were items that, while not an error or an exception, could cause problems and should be reviewed.

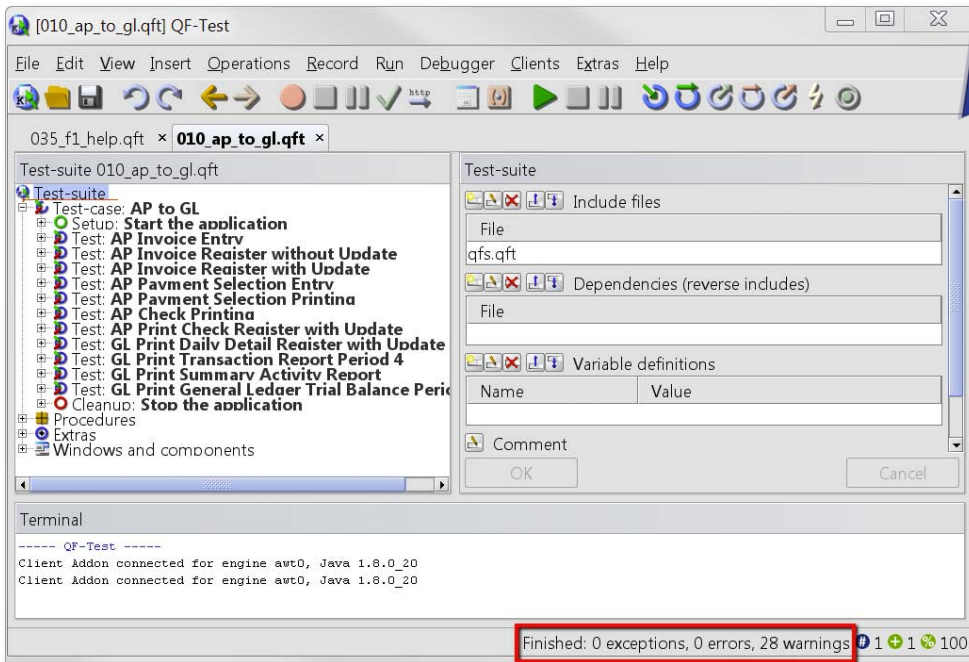


Figure 3. Successful Test Suite

To Test or Not to Test, That was the Question

With manual testing, there is the age old trade-off of resource costs against the quality gained. Deciding when to test was as much an art as a skill. However, with QF-Test's automated tests, we can run our tests as often as we like, kicking them off as a side-effect of a successful build, and have the results ready to view first thing the next morning. One benefit of this that often goes overlooked is that these same tests now work as Regression Tests. That means that we can run them on every build, regardless of what the development team has changed, and even detect side effects that may arise in unrelated areas of the product. In other words, we can make sure that we didn't break something in one place by fixing a defect in another.

Many Happy Returns on our Investment

Automated testing is not for everyone – there can be a considerable up-front cost in training and in developing the automated tests needed to cover your program adequately. Nothing is free. At BASIS, it is worth it. We do nightly automated builds and run the QF-Test tests on the result, only requiring human intervention when the test detected a defect. We call that "peace of mind," and it saves us money in the end. The sooner we detect a problem, the less expensive it is to fix, the faster we can release our products, and the sooner our developers and testers can move on to other important tasks. ■

"In particular, the testing of GUIs is more complex than testing conventional software, for not only does the underlying software have to be tested but the GUI itself must be exercised and tested to check for bugs in the GUI implementation. Even when tools are used to generate GUIs automatically, they are not bug free, and these bugs may manifest themselves in the generated GUI, leading to software failures."

Atif Memon, et al., Using a goal-driven approach to generate test cases for GUIs in 'Proceeding of the 21st International Conference on Software Engineering,' pages 257-266, IEEE Computer Society Press, 1999



Bleeding Heart Computer Security

Every day the Internet is a battlefield where [cyber security black hats and white hats](#) fight for control of your computer. Some days the good guys win. Some days the bad guys win. And some days it isn't clear that anybody wins. But one thing is for sure, computer security is something that can only be ignored with great risk, and it is likely that it will only become more critical to you and your business as time marches on.

BASIS is committed to monitoring and responding to computer security issues as they are detected. Occasionally a major security issue is announced that requires immediate action – such as the case with the [Heartbleed Vulnerability](#).

Heartbleed Vulnerability

The Heartbleed Vulnerability was a major security issue that was announced in April 2014 as “CVE-2014-0160,” where CVE ([Common Vulnerabilities and Exposures](#)) is the Standard for Information Security Vulnerability Names maintained by the MITRE Corporation.



Heartbleed was essentially a security hole in versions 1.0.1 through 1.0.1f of an Open Source encryption library called OpenSSL. If an attacker could open a connection to a service using an affected version of OpenSSL, the security hole allowed the attacker to extract random data from a memory buffer that might contain sensitive data such as passwords and encryption keys.

“The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.” [heartbleed.com](#)

When the news came out about Heartbleed, our immediate concern at BASIS was to determine what impact Heartbleed might have on customers using our product line. Heartbleed required an immediate investigation and an immediate response.

Our investigation showed that BBj® was not affected. The only BBj component that used OpenSSL was an ODBC client for Windows, which has both a 32-bit and a 64-bit version. At that time the 32-bit version was using OpenSSL 0.9.x, which was



By Dan Christman
Software Engineer



By Jerry Karasz
Software Architect

unaffected by Heartbleed, but the 64-bit version was using OpenSSL 1.0.1c and was potentially at risk. Further investigation showed, however, that the 64-bit ODBC client was not vulnerable to the Heartbleed issue because Heartbleed only affected software that would accept incoming socket connections.

The investigation also showed that PRO/5® and Visual PRO/5® used OpenSSL for SSL sockets, but that the OpenSSL version being used was 0.9.x, which was unaffected by Heartbleed. Having determined from our investigation that no action was needed on BASIS' part, and that all was right with the world, we announced [our results](#) and went back to our development tasks secure in the knowledge that Heartbleed was not our problem.

Reverse Heartbleed Vulnerability

And then came the Reverse Heartbleed announcement. Reverse Heartbleed was a security hole similar to Heartbleed that allowed a malicious server that accepted a connection from a vulnerable client to extract random data from the client's memory buffer (that is why it is referred to as the "reverse"). Reverse Heartbleed could only be exploited by tricking a vulnerable client into connecting to a malicious server, a situation that was not as common as the Heartbleed vulnerability. Back to work to investigate the BASIS product line yet again.

The investigation showed that the 64-bit BBJ ODBC client for Windows was the only product that offered any possible vulnerability to [Reverse Heartbleed](#). Since it was just barely possible that a malicious server could steal sensitive information by tricking a client into using a BBJ 64-bit ODBC client to connect to it, we upgraded the 64-bit ODBC DLL to the latest OpenSSL version. We released an updated 64-bit ODBC DLL for those who might need to patch an older version of the BBJ 64-bit ODBC client for Windows without upgrading to the latest version of BBJ, and created a knowledge base article [Heartbleed Fix for 64-bit Windows BBJ ODBC Driver](#) to help customers understand the issue.

Proactive Updates

Even though PRO/5, Visual PRO/5 and the 32-bit BBJ ODBC client for Window were never affected by Heartbleed or Reverse Heartbleed, BASIS upgraded the OpenSSL library that those products use to the latest version that contained the official fix for those vulnerabilities.

Besides the potential language impact, Heartbleed brought an operating system compliance cost. BASIS' [Transformer AMI](#) (Amazon Machine Image) includes a version of OpenSSL, and Amazon Marketplace insisted on a full upgrade to OpenSSL version 1.0.1g. Shortly thereafter, they required a second compliance upgrade to OpenSSL version 1.0.1h.

In today's compliance-driven world, it is important to remain current with the most secure versions of the software upon which your business relies. BASIS' Transformer AMI has a compliant OS, and the newer versions of BASIS products ship with a current version of OpenSSL that contains the official fix for both vulnerabilities.

Summary

The BASIS product line was never affected by the Heartbleed vulnerability because its only affected component could not accept incoming socket connections. The 64-bit ODBC DLL that had a vulnerable version of OpenSSL was affected by Reverse Heartbleed in that it cannot accept incoming connections.

Computer security is a constant battle. Everyone can make a mistake, as the developers of the OpenSSL library did. The important thing is that we all remain vigilant and on the lookout for as many of these problems that we can and work to address them as quickly as possible.

Remember, if your applications use an OpenSSL library with version 1.0.1 through 1.0.1f (inclusive) for socket communications, you should review the information about Heartbleed and Reverse Heartbleed and investigate your programs for yourself. You may have created a security issue of your own. Meanwhile, we at BASIS will continue to watch for potential security issues in our product line and work to keep you informed and up to date. Keep your BASIS products updated to be secure! ■



For more details, refer to the knowledge base article [Heartbleed Fix for 64-bit Windows BBJ ODBC Driver](#)



Replication Redux

The “Guiding Principle of Replication” at BASIS is to use the fewest resources necessary to continuously back up the data as requested by the user and remain as current as those resources will allow. Clearly, some resource use is required, but the impact on user operations should be as small as possible and be imperceptible to the user group. In pursuit of this goal, BASIS added these improvements to shrink replication resource usage:

- Reduced the number of open files on both the target and the source
- Optimized the copying of small files and large string files from the source to the target

These changes greatly improved the efficiency of a replication job with large numbers of files to copy from the source to the target, such as when there is no initial rsync or when creating or changing a large number of string files. This article takes a closer look at these improvements.

Caching File Opens on the Target

The first improvement was to tighten control over the number of files open on the replication target. A replication target needs a file open in order to make such changes as writing or removing records. However, replication operations are a stream and encompass changes to potentially many files. It is much too expensive to open and close a file every time there is a file operation. After all, the user might be in the process of adding a million records to the same file, and opening and closing the file a million times (once for each record) is much too slow.

To resolve the issue, we keep a cache of recently used files on the replication target; the first operation on the file will open it and subsequent operations will find it already open and ready to modify. If the file has not had any operations for a couple of minutes, we go ahead and close it in order to avoid having too many files open. Unfortunately, this does not help if users were modifying many files at the same time. If users modified a thousand files within a couple of minutes, then the replication target could open all thousand files at the same time, which would lead to running out of file handles. BASIS resolved this by adding a restriction on the total number of open files in the cache. In addition to closing files when they have not been used for a couple of minutes, the total number of open files is limited to prevent overwhelming the replication target with open files even if there are near simultaneous changes to many different files on the source.

Limiting Open Files on the Source

BASIS also made changes to minimize the number and duration of open files on the replication source. Unlike the target, the replication source only needs to open files in order to check for OS-level changes and to copy files to the target. We record simple data file modifications directly to the replication log and send them on to the target without needing to open the file. However, when we detect OS-level changes to a file, we need to open the file to check the contents against the target and copy it when necessary. Originally, we would open all of the source files in order to check for changes and if we required a new copy of a file we would keep that file open until



By Chris Hardekopf
Software Engineer

the copy completed. Unfortunately, this could lead to problems. It could cause each file to remain open for an extended period of time, using up open file handles and preventing it from being deleted.

We changed how the copy works so that instead of keeping the file open until it was copied, we immediately close the file and add the name to a queue. When we are ready to actually copy the file, we attempt to open it again just for the duration of the copy. This means that only a few files are open at a time and the user is able to delete the file while it is waiting for the job to copy it since it will only be open for the minimum time necessary for the check and thereafter for the actual copy.

Copy Whole Files

Copying files from the replication source to the target is normally performed a block at a time for string files, or a record at a time for data files. This lets us efficiently copy large files as well as allow file modifications during the copy process. However, if the file is small enough, we now copy the whole file in a single operation. It is much more efficient to simply load the entire file into memory and send it to the target. This process avoids extra communication between the source and target about the file, minimizing the amount of time the file is open on the replication source. However, copying large files in this manner would take too much time and use too much memory and network bandwidth in loading the entire file into memory and sending it to the target.

Add Checksums to Minimize Traffic

Occasionally, a user will put a large string file into a replication job, requiring the replication source to copy the file to the target. If the file does not exist on the target, the job must copy the entire file. However, sometimes the large file already exists on the target and users are only modifying the source file in relatively small ways. For example, when users append to log files and only change the end of the file.

In order to handle such cases better, BASIS changed large string file replication to get checksums for blocks of the file that already exist on the target and only copy the parts of the file that have different checksums or that do not already exist on the target. Now, instead of just copying the file from scratch as it used to, the replication target iterates through the target file gathering a list of checksums that it can send back to the replication source. The replication source then iterates through the source file it is copying, comparing the checksums in order to determine which parts of the file have actually changed. The replication job only needs to send the changed parts of the file over the network to the replication target. This process requires that the replication target read the target file, and the replication source still needs to read the entire source file, but it has the potential to greatly reduce network traffic and speed up the file copy for minor changes (such as appending) to large string files.

Summary – Faster, More Robust, More Efficient

At BASIS, our belief in the concept of continuous improvement made these recent changes to replication an easy target, utilizing the feedback from the community and our own research helped us make the process as efficient and unobtrusive as possible. We look forward to finding more ways to improve the performance and reduce resource usage into the future. ■



For more information, refer to:

- [Replication Introduction](#) in the online documentation
- [Anatomy of a Replication Job](#) in The BASIS International Advantage

**Sit back and enjoy a
30-minute presentation
with BASIS!**

b.g.basis.com/javabreak



AddonSoftware's Digital Dashboard Takes Off



*By Carla Johnson
Software Developer*



*By Christine Hawkins
Software Developer*

Digital dashboards provide a graphical view of business data, allowing anyone, especially owners and management executives, to quickly and easily make better informed decisions. To capitalize on the value of visual data presentation, the AddonSoftware® development team undertook the task of offering the value-added reseller (VAR) community a representative, well-rounded sampling of ERP widgets that would both pique prospects' interest as well as supply a prototype that VARs could use for their own vertical development. To that end, AddonSoftware by Barista® version 14.0 not only debuts a fully functional dashboard (**Figure 1**), but also provides a solid foundation for customization.

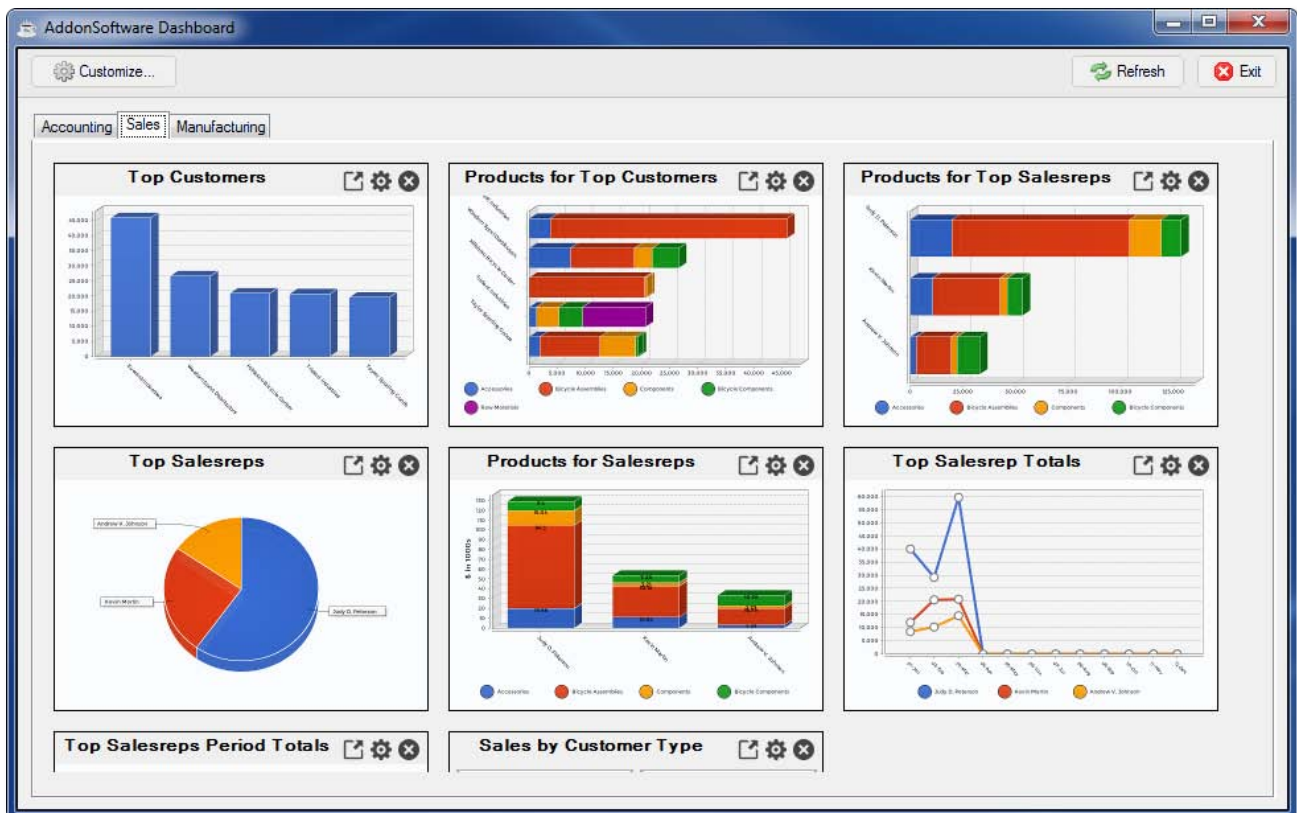


Figure 1. AddonSoftware Dashboard showing widgets in the sales category

The AddonSoftware Dashboard includes examples of most of the new BASIS Dashboard Utility's widget types, graphically displaying key data in an easily digestible manner. In addition, version 14.0 showcases the dashboard's flexibility with an embedded widget (**Figure 2**) on the Accounts Receivable Customer form. In the AddonSoftware tradition, all the code that makes this magic happen is available to the VAR community.

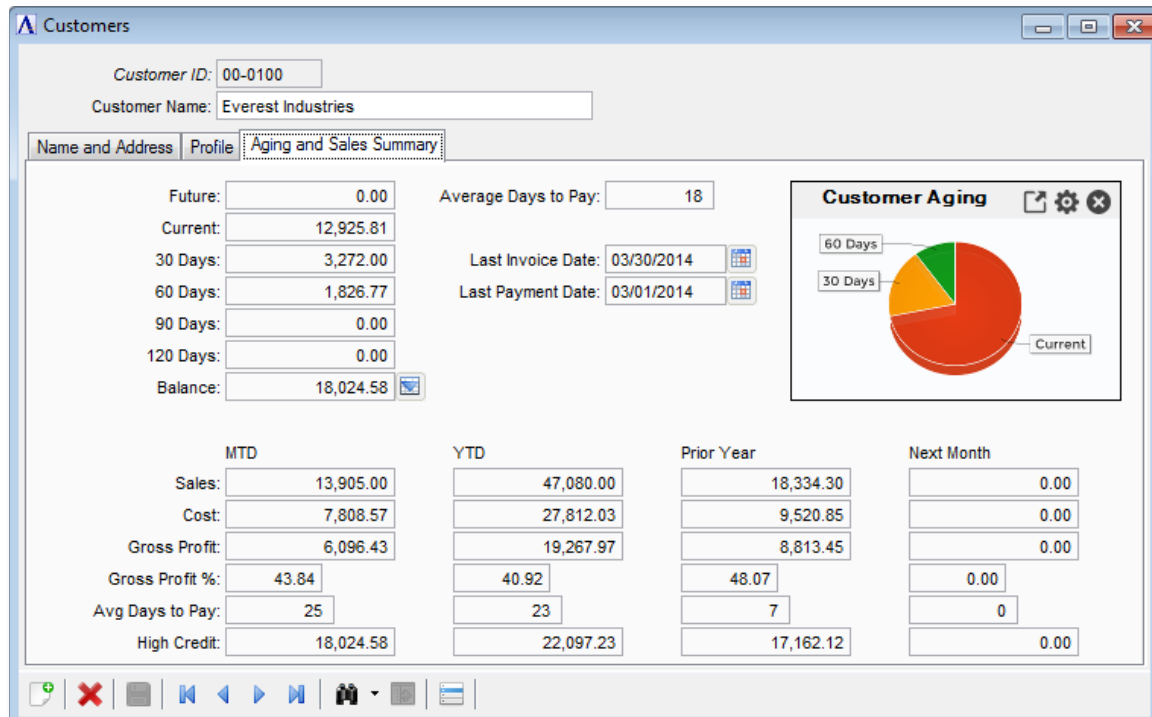


Figure 2. Customer form contains an embedded pie chart showing aged balances

This article unearths what we learned during our development cycle. Read about our design process and the aspects of development that were both easy and challenging, and get a peek “under the hood” at the technical details.

Getting Started

The AddonSoftware team was excited to participate in the development of the new BASIS Dashboard Utility and have the opportunity to work closely with BBj® engineers to help shape the dashboard's functionality and direction. As the BBj team developed the utility, we molded the mechanics to AddonSoftware – always keeping in mind our goal of assisting both VAR developers and resellers. AddonSoftware inherits all the functionality of the Dashboard Utility, including the ability to pop out individual widgets for a zoomed-in view, save or email a widget image, manage refresh options, and customize the dashboard layout.

With so much inherited functionality, we were able to focus on how to make the best use of the Dashboard Utility within AddonSoftware and the Barista Application Framework. We started by brainstorming ideas on which widgets to include in our initial release. To meet our goal of delivering a dashboard that is both customer-facing and a prototype for VARs, we identified some initial requirements. Widgets needed to

- Offer an eye-catching display
- Provide an effective presentation of data relevant to a prospect's business
- Reference data that would lend itself to a graphical depiction
- Be appropriate for the data being collected

As we considered these requirements, AddonSoftware's General Ledger and Sales Analysis data tables came to the fore as a "widget-rich" environment. Our design plan included a broad selection of widgets to benefit both prospects and VARs as shown in **Figure 3**.

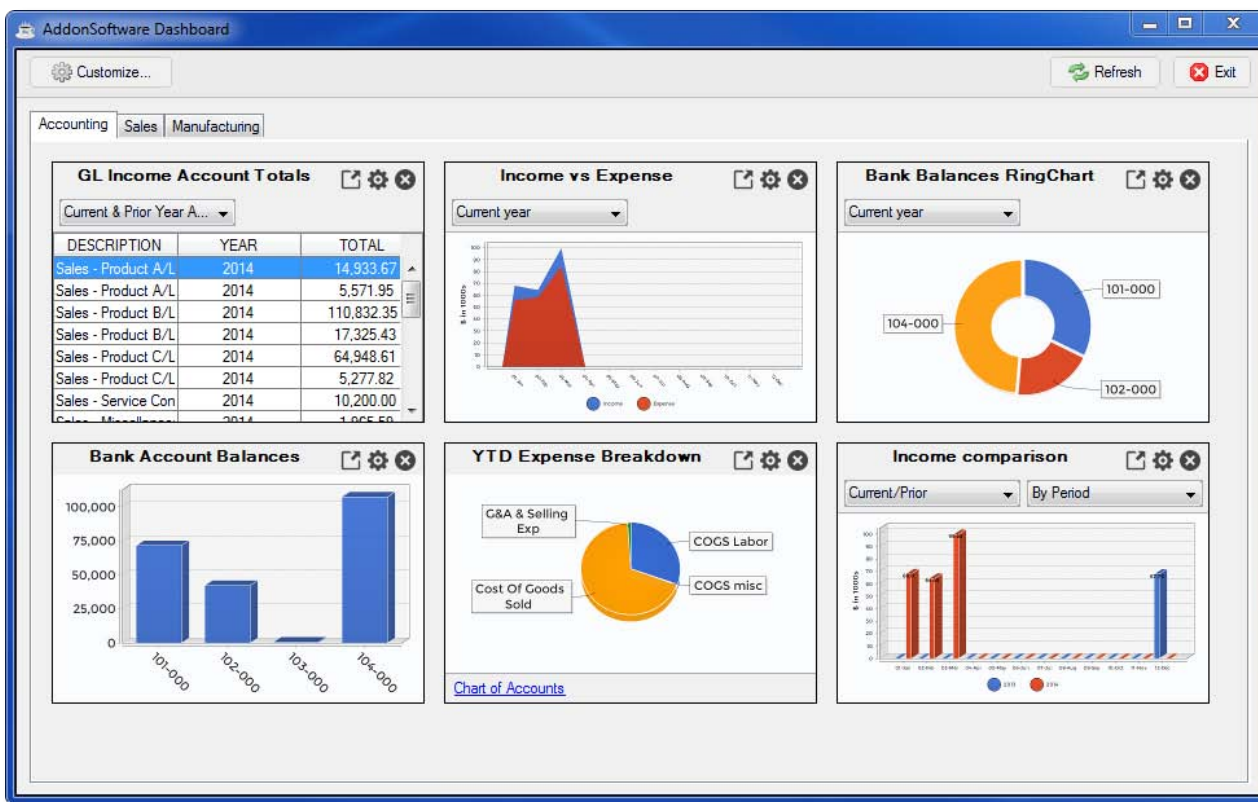


Figure 3. AddonSoftware Dashboard showing widgets in the 'Accounting' category

Once we defined the core set of widgets, we turned our efforts toward designing a process to launch the dashboard in accordance with the Barista security model so the availability of widgets would tie to the user's security role. We also considered AddonSoftware's modular structure so that the dashboard would only create widgets for installed modules.

Understanding the Dashboard Utility

Just as the AddonSoftware Dashboard has laid the foundation for VARs and resellers to demo the new dashboard and customize or expand on it, the Dashboard Utility paved the way for the AddonSoftware team. Not only could we refer to the [Dashboard Utility Overview](#) in the online Help to learn about the various aspects of the utility, but we also got a boost by having a functional demo dashboard with access to the source code. And all of these resources – the online documentation, the BBj Demo Dashboards, and the Addon Dashboard – are available to anyone developing with BBj!

We began by taking the demo dashboard for a test drive to acquaint ourselves with its function, capabilities, and available widget styles. We analyzed the underlying code, and found that we could leverage pivotal routines – specifically, the logic for constructing the dashboard, its categories, and its individual widgets.

The methods for constructing the various widgets and setting their properties are consistent and intuitive, so the code was easy to understand and propagate. Furthermore, we found that because the Dashboard Utility provides defaults for colors, fonts, etc., we could create a widget very quickly with just a few parameters. We could set and/or change many additional properties if we chose, but didn't need to worry about a myriad of details to get up and running.

Implementing the Dashboard in AddonSoftware

Since we could borrow the core code from the demo dashboard to get the basics for AddonSoftware's dashboard in place, we were free to focus on the application-specific challenges. Some of these challenges were specific to widgets while others applied to the overall design and, once solved, would be in place for the benefit of others doing dashboard development.

Look and Feel

In terms of overall design, we had to think about the general look and feel of the dashboard. While it was tempting to use different types of widgets, colors, and fonts, and to experiment with rendering the data as a flat vs. 3D graphic, we opted to let the utility's defaults set the theme. This decision made our job easier and also provided a more consistent look.

Launch

Since we needed the ability to launch the dashboard either inside the Barista MDI or via the browser user interface (BUI), we added code to update the progress meter in Barista's menu panel, and perform other miscellaneous initialization tasks to facilitate a BUI launch.

Security

Barista's security is tied to menu items; therefore to address security considerations, we created hidden menu items for each widget (**Figure 4**), not just for the dashboard as a whole. This gave us the infrastructure to set Barista security options on a per-widget basis to tailor each user's version of the dashboard to their security permissions (**Figure 5**). Without the granularity provided by assigning menu items to each widget, security could only be applied at the level of the dashboard as a whole – a user could either access all of the dashboard or none of it. Clearly, that would not have been an optimal solution.

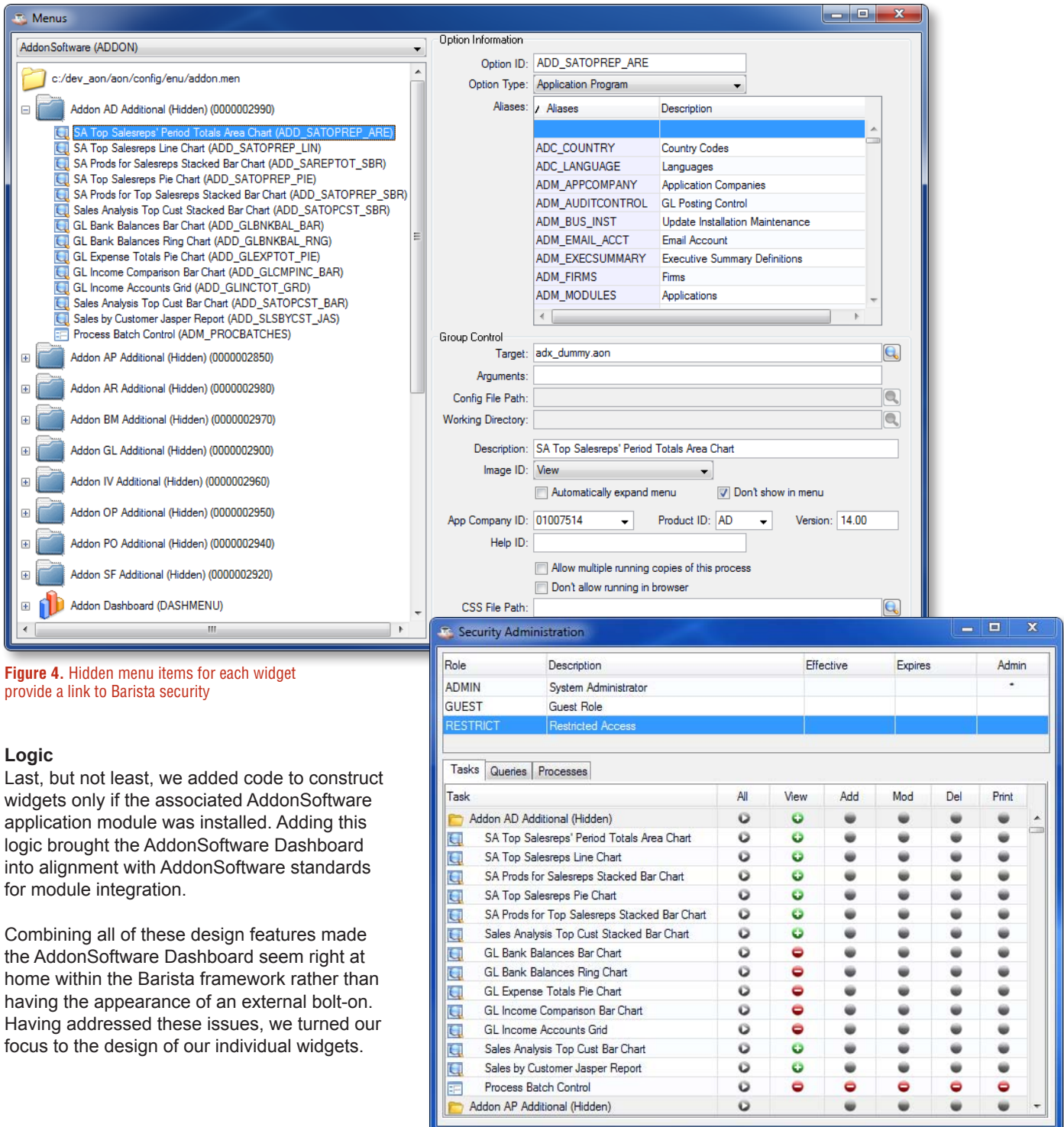


Figure 4. Hidden menu items for each widget provide a link to Barista security

Logic

Last, but not least, we added code to construct widgets only if the associated AddonSoftware application module was installed. Adding this logic brought the AddonSoftware Dashboard into alignment with AddonSoftware standards for module integration.

Combining all of these design features made the AddonSoftware Dashboard seem right at home within the Barista framework rather than having the appearance of an external bolt-on. Having addressed these issues, we turned our focus to the design of our individual widgets.

Figure 5. Barista Security Administration controls access to widgets based on user role

Setting Design Standards

At the widget level, we found that even though we had to revisit certain design questions with each widget, the process became easier after the first time through, and was even faster by establishing some standards.

The first question in designing our widgets was, “Which of the many widget types would best depict the data we wanted to display?” The answer was largely an educational step, and the team went to the Internet to learn about charting. For example, consider the two representations of sales rep data shown in **Figure 6** and **Figure 7**. Shown as a line chart, the individual sales amounts for each rep display clearly. The same data shown as a stacked area chart places the reps' sales amounts atop each other, so you also have a visual of total sales.

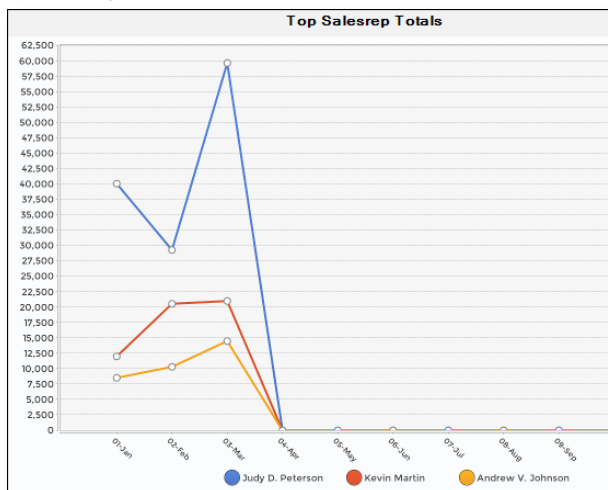


Figure 6. Line chart presents individualized data for Top Salespersons from Sales Analysis

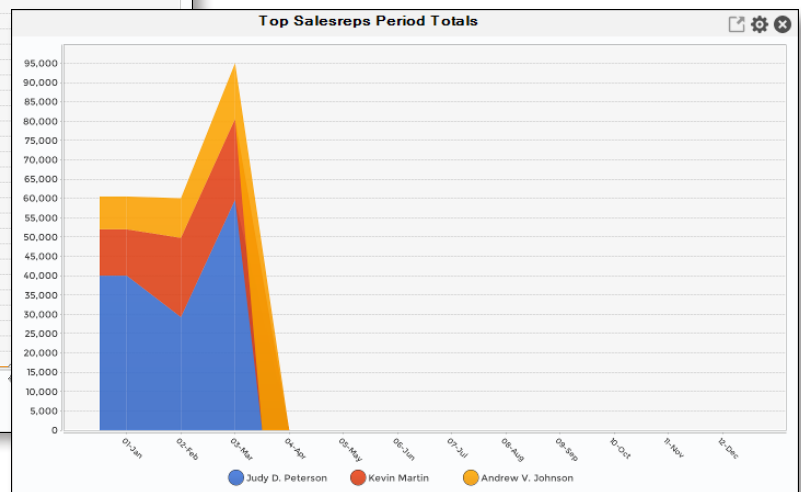


Figure 7. Top Salesrep total sales by period in a Stacked Area Chart calls out the totals for each period

It is also important to understand that different widgets require different recordset structures, so we had to decide on the widget type before we set about writing the stored procedure (SPROC). **Figure 8** shows examples of the recordsets needed for a pie chart vs. a stacked area chart.

SALESREP	TOTAL
Judy D. Peterson	128912
Kevin Martin	53460.5
Andrew V. Johnson	33237.5

SALESREP	PERIOD	TOTAL
Judy D. Peterson	01-Jan	40008
Judy D. Peterson	02-Feb	29264
Judy D. Peterson	03-Mar	59639.5
Judy D. Peterson	04-Apr	0

Figure 8. Recordset for sales rep pie chart (left) and stacked area chart (right)

Since SPROC's are written with BBj code, we had the freedom to write them using either SQL or native file access. We wrote a few selected SPROC's with both kinds of access, then REM'd out one or the other and ran traces and/or called the SPROC's from within the Enterprise Manager to compare the performance. In general, if the desired data came from only one or two tables and was already in a normalized form and adequately indexed, SQL worked great. Otherwise, we found we could get our return recordset more quickly using native file access. That principle made it faster to design and code the remaining SPROC's.

Adding Filters

Another widget-specific consideration was whether to add filters so the user would have options for tailoring the result. For example, the Sales category contains a widget showing the AddonSoftware Accounts Receivable Drilldown Sales Report. When run stand-alone from the menu, the user establishes the month and year for the report. In the dashboard version, the widget initially appears based on defaults set out in the code, and the user can then select the period of their choice from the month/year filters (**Figure 9**).

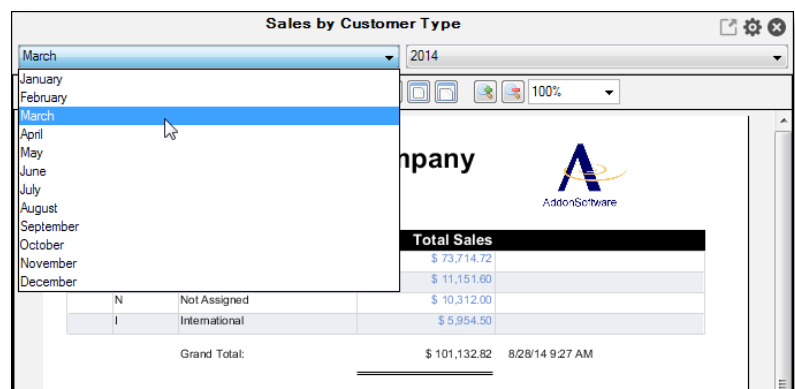


Figure 9. Filters allow users to change the widget data dynamically

Delivering our Final Product

At this point in the development, we had sailed over most of the hurdles, resulting in minimal ramp-up time for VARs wanting to demo or modify the AddonSoftware Dashboard. By following a process similar to our undertaking, a VAR's step into the new world of graphical data display should be straightforward. The general process is the same as any AddonSoftware development.

1. Define your goal.
2. Familiarize yourself with the user interface.
3. Review existing code logic pertaining to your goal.
4. Follow the examples provided when making your custom logic.
5. Consult the BASIS [Dashboard Utility](#) documentation, as needed.

Read on for some of the more technical details of our implementation.

Dissecting the Components

As an overview, the BASIS Dashboard Utility provides the underlying widget and dashboard objects and contains a number of components. Refer to the links at the end of this article for more information about the Dashboard Utility that is published in this issue and the online documentation.

Barista handles framework functionality such as security, document warehousing, STBLs, sysinfo, localization, and masking. AddonSoftware leverages these primarily via a program named `adx_aondashboard.aon`, a launcher program that contains all of the logic to interface the AddonSoftware data with the BBJ and Barista components. To help with customization efforts, this program includes examples of the majority of widget types, each of which utilize calls to SPROCs to collect data. Its easy-to-follow structure lends itself to customization.

The flow of `adx_aondashboard.aon` is essentially this:

1. Initialize various processes and variables, including a check for BUI.
2. Create the dashboard.
3. For each category, create the category tab control.
4. For each widget, if security allows it and the associated module is installed, create the widget.

Creating the dashboard and its categories is very straightforward – just two lines of code each. To illustrate, **Figure 10** shows the code snippets that create the dashboard itself and the accounting category.

```
rem =====
create_dashboard:
rem =====
declare Dashboard aonDashboard!
aonDashboard! = new Dashboard("Addon", Translate!.getTranslation("AON_ADDONSOFTWARE_DASHBOARD"))
return

rem =====
create_acct_tab_and_widgets:
rem =====
declare DashboardCategory acctDashboardCategory!
acctDashboardCategory! = aonDashboard!.addDashboardCategory("Accounting", Translate!.getTranslation("AON_ACCOUNTING"))
```

Figure 10. The code that creates the Dashboard and the Accounting category

Initially we borrowed the code for creating individual widgets from the BASIS Demo Dashboard, altered it for AddonSoftware, and then wrapped it in the logic to create the widget based on the user's security and whether or not the requisite module is

installed (**Figure 11**). Note also that the user-facing text – the dashboard title, category names, widget titles, filter contents, etc. – are localized to display the appropriate text for the user's locale/language.

```
rem =====
rem --- create SA Top Salesreps pie chart
rem =====

dashboard_menu_id$="ADD_SATOPREP_PIE"
gosub get_security
if allow_widgets="Y" and installMap!.get("SA")="Y"
  name$ = "SATOPREP_PIE"
  title$ = Translate!.getTranslation("AON_TOP_SALESREPS")
  previewText$=Translate!.getTranslation("AON_TOP_SALESREPS_DISPLAYED_IN_A_PIE_CHART")
  previewImage$=preview_path$+"satoprep_pie.png"
  chartTitle$ = ""
  flat=0
  legend=0
  numSlices=8
  connectString$=aon_url$

  rem --- params for calling SPROC
  year$ = proc_date$(7,4)
  num_to_list$ = "5"

  sql$="CALL SATOPREP_PIE ('"+firm_id$+"', '"+year$+"', '"+num_to_list$+"', '"+mask$+"
:      ", '"+barista_wd$+"')"
```

```
toprepPPieChartDashboardWidget! = salesDashboardCategory!.addPieChartDashboardWidget(
:   name$,title$,previewText$,previewImage$,chartTitle$,flat,legend,connectString$,sql$)
toprepPWidget! = toprepPPieChartDashboardWidget!.getWidget()
toprepPWidget!.setFontScalingFactor(0.45)

gosub update_meter
endif
```

Figure 11. Widget construction code wrapped with tests to check security and application installation

Abbr.	Type
ARE	Area chart
BAR	Bar chart
GRD	Grid
JAS	Jasper widget
LIN	Line chart
PIE	Pie chart
RNG	Ring chart
SAR	Stacked area chart
SBR	Stacked bar chart
SPB	Stacked % bar chart
XYC	XY chart

Figure 12. Three-character widget types used in Addon Dashboard naming conventions

To simplify widget creation, we developed a naming convention for widgets. We then broadened the convention to add consistency across the various components (SPROCs, Barista menu ID's, widget names, thumbnail images, and filter selection events). Per AddonSoftware standards, AddonSoftware Dashboard names include a two-character module ID and descriptive acronym. For clarity, the last four characters are an underscore followed by a three-character widget-type abbreviation as shown in **Figure 12**.

Embedding a widget in a form like we did on the 'AddonSoftware Customer' form (**Figure 2**) uses code that is very similar to adding a widget to the dashboard, except we used the EmbeddedWidget classes as shown in this excerpt from the 'After Show (ASHO)' callpoint (**Figure 13**).

```
rem --- Create/embed dashboard to show aged balance

use ::dashboard/widget.bbj::EmbeddedWidgetFactory
use ::dashboard/widget.bbj::EmbeddedWidget
use ::dashboard/widget.bbj::EmbeddedWidgetControl

rem --- pie
name$="CUSTAGNG_PIE"
title$ = Translate!.getTranslation("AON_AGING", "Customer Aging",1)
chartTitle$ = ""
flat = 0
legend=0
numSlices=6
widgetHeight=ctl2!.getY()+ctl2!.getHeight()-ctl1!.getY()
widgetWidth=widgetHeight*widgetHeight*.5

agingDashboardPieWidget! = EmbeddedWidgetFactory.createPieChartEmbeddedWidget(name$,title$,chartTitle$,
:   flat,legend,numSlices)
agingPieWidget! = agingDashboardPieWidget!.getWidget()

agingPieWidget!.setDataSetValue(Translate!.getTranslation("AON_FUTURE", "Future",1), 0)
agingPieWidget!.setDataSetValue(Translate!.getTranslation("AON_CURRENT", "Current",1), 0)
agingPieWidget!.setDataSetValue(Translate!.getTranslation("AON_30_DAYS", "30 Days",1), 0)
agingPieWidget!.setDataSetValue(Translate!.getTranslation("AON_60_DAYS", "60 days",1), 0)
agingPieWidget!.setDataSetValue(Translate!.getTranslation("AON_90_DAYS", "90 days",1), 0)
agingPieWidget!.setDataSetValue(Translate!.getTranslation("AON_120_DAYS", "120 days",1), 0)
agingPieWidget!.setFontScalingFactor(1.2)

agingPieWidgetControl! = new EmbeddedWidgetControl(agingDashboardPieWidget!,childWin!,
:   ctl1!.getX()+ctl1!.getWidth()+50,ctl1!.getY(),widgetWidth,widgetHeight,$$)
agingPieWidgetControl!.setVisible(0)
```

Figure 13. Callpoint code in Addon's Customer form constructs an embedded pie chart widget

Comparing **Figure 11** with **Figure 13**, you'll notice numerous points of similarity, including the use of the getTranslation method to localize the user-facing text in accordance with AddonSoftware's multilingual capabilities.

Tips on Customizing

VARs wishing to customize the AddonSoftware Dashboard already have a roadmap in place to follow. We've done the heavy lifting so developers don't have to! Here are some things to keep in mind if you want to build your own dashboard or modify the standard.

- Customizations should pay attention to the handling of security as demonstrated in `adx_aondashboard.aon`.
- Making use of the same/similar naming conventions that AddonSoftware implemented in the standard product will make it easy to identify/locate the various components (SPROCs, Barista menu ID's, thumbnail images, etc.).
- Begin by reviewing the data underlying the widget you want to create so that you can decide on the right widget for the job and whether you'll want to use SQL or native file access to build your recordset.
- Don't forget that each SPROC needs to be defined in Enterprise Manager. AddonSoftware does this with the `adx_bui1dsproc.aon` utility that runs with the first launch of AddonSoftware (via Barista's Auto-Launch mechanism). VARs can create a similar auto-launch process to make sure SPROCs are re-defined after new installations or upgrades.

Summary

The BASIS Dashboard Utility throws the data visualization doors wide open and AddonSoftware's dashboard implementation gives VARs not only a ready-made solution for demonstrating graphical capability in the application, but also a great tool to use as a springboard for customization. ■



- Refer to the following resources for deepening your understanding of AddonSoftware's Digital Dashboard
 - [Dash Boredom With the Dashboard Utility](#)
 - [Easier Decision Making With the Dashboard Utility](#)
 - [Dashboard Utility Overview](#) in the online documentation
- Download and run the [code samples](#)





Test for Success With BBJ Unit Test

You finally finished writing your BBJ® application, and it's time to put it out in front of your end users. Or maybe it's not – how do you know when software really is ready for public consumption? The answer may be “*When the scheduled release date arrives,*” or “*When the boss says it's time,*” or maybe even “*When we've put it through its paces and there are no more significant bugs to fix.*” Sure, we all know that any good software development life cycle includes time for testing, starting with unit testing and ending with some form of system or acceptance testing. But how much time and money can you afford to invest in testing? How do you use your limited testing dollars to get the most bang for your buck?

Common sense says that the earlier you find a bug the easier and cheaper it is to fix it, and our experience at BASIS supports this conclusion. But this is only helpful if you don't have to spend large amounts of money or time in order to find those bugs early. So what you need is a relatively cheap tool to help you find bugs in your BBJ code as early as possible. How about finding bugs as a code change creates them?



By Jerry Karasz
Software Architect



Sebastian Adams
Software Developer

While you are creating a new BBJ application, wouldn't it be nice if you could just push a button and find out what works and what doesn't? What would you give to have a series of tests that you can run any time, over and over, quickly turning out a clear and concise report of which passed and which failed? **Unit testing** offers an opportunity for just such a return on even a small investment. But unit tests are not free – somebody has to write every one of them. The return comes once you have written your unit tests. You can run them over and over again, not only to find bugs as you are developing your code, but even afterwards to find out that you broke something in your code with that latest bug fix.

Unfortunately, unit testing has never before been an option for BBJ developers – until now. BASIS is proud to announce the first step in providing a BBJ Unit Test framework that you can use to develop and run unit tests for your BBJ code – the BBJ Unit Test Eclipse plug-in. For those of you familiar with CppUnit, JUnit, or any of the other **xUnit** frameworks, you will see a lot that you are familiar with here.

Preparing for Unit Testing

Although this plug-in is still in its infancy, it does provide the basis for a good unit testing strategy. Let's look at where to get it and how to use it.

Getting the Plug-in

Go to the BASIS Eclipse page and choose the **BBjUtilities** "Release" URL. Install according to the "**BASIS AND THIRD PARTY PLUG-INS**" instructions. Eclipse will download any newer version when it checks for updates.

Creating Unit Test Code

Suppose you have a BBj project named **MathProj** that contains a BBj code file named **MathOperations.bbj** as shown in **Figure 1**.

MathOperations is a simplistic class and offers just enough methods to be useful but not enough to complicate this example. It has methods defined to do multiplication, addition, subtraction, and division. But does it do them correctly? To find out, let's set up some unit tests.

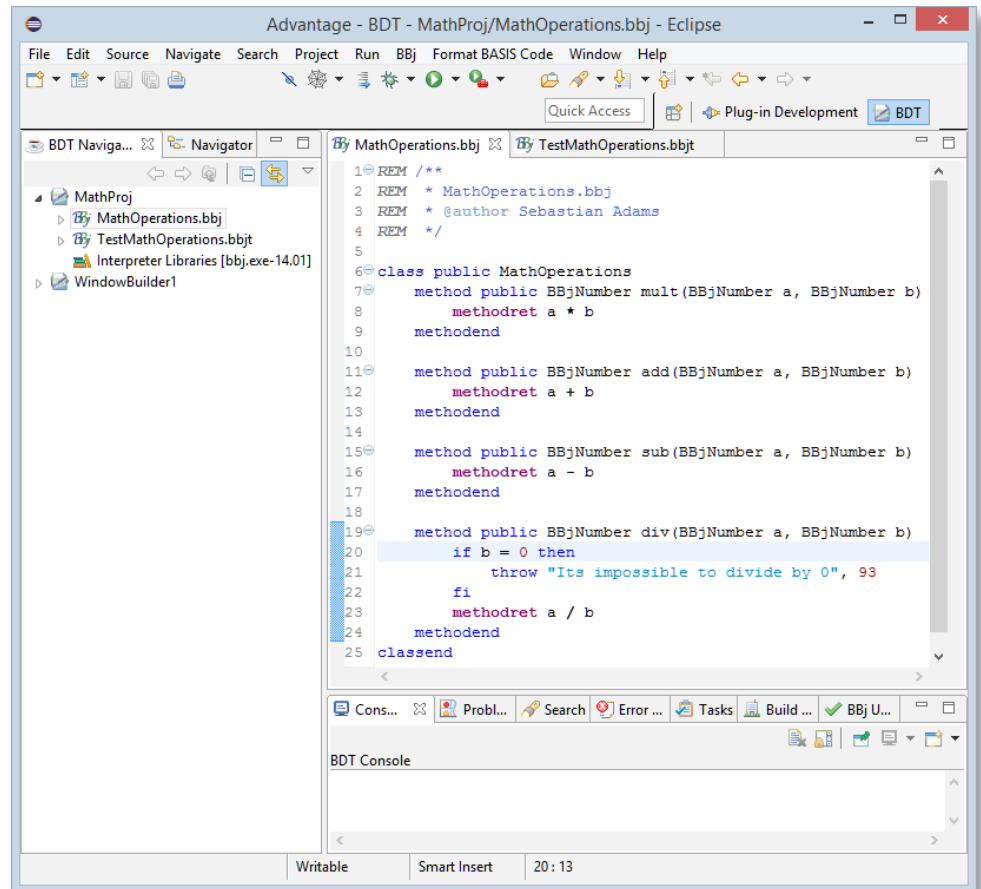


Figure 1. A 'BBj Project' that contains a simple mathematics class

What Exactly is a Unit Test?

In BBj Unit Test, unit test is a method on a class that returns nothing (it has a void return type) and announces its success or failure through the special **Assert** methods it calls (more on the **Assert** methods later). In order for the BBj Unit Test framework to recognize your method as an official "unit test" and be able to run it, your method also needs to be flagged as a unit test method by making the line of code immediately preceding the method declaration an annotation remark, **REM @Test** (case insensitive). For an example of a simple unit test method, see **Figure 2**.

Where do I put my Unit Test Code?

Unit test methods such as **addTest()** must be part of a unit test class. We could create a class directly in the **MathOperations.bbj** file to hold all of our unit test methods, but once **MathOperations** and our tests get large, this will become harder to manage, and running our tests in Eclipse will conflict with running the regular **.bbj** file. Besides, it is considered poor programming style. For these reasons, the BBj Unit Test plug-in requires test classes to be in a separate file in the same BBj project with a specific extension: **.bbjt**. We also recommend you name the test class and file something intuitive so that the relationship between the file being tested and the unit test file is obvious. For our example, let's name our unit test class **TestMathOperations** and our unit test file **TestMathOperations.bbjt**, and put **TestMathOperations.bbjt** in the same **MathProj** project (see how this looks in the BDT Navigator view shown in **Figure 1**). Starting the unit test class name and filename with "Test" is a convention that we recommend, but it is not required.

How do I Structure my Unit Test Class?

In this example, we will probably need an instance of the **MathOperations** class in every unit test method we write, because that is what we are testing. We could allocate a **MathOperations** variable in every unit test method, but that seems redundant and a lot of work. We could put in a field that is accessible to every unit test method such as **field private MathOperations mathOperations!**, and then our class would look something like **Figure 3**.

```
REM @Test
method public void addTest()
    Assert.Equals(#mathOperations!.add(8,8),16)
methodend
```

Figure 2. An example unit test method

```
use ::MathOperations.bbj::MathOperations

class public TestMathOperations
    field private MathOperations mathOperations!

    rem @Test
    method public void addTest()
        Assert.Equals(#mathOperations!.add(8,8),16)
    methodend
classend
```

Figure 3. A partial BBj Unit Test class, **TestMathOperations**

But that won't work – we need a way to initialize **field mathOperations!** to an instance before we can use it, and that needs to be done exactly once before we start running our tests.

If we just had some way to run some code before we executed any of our unit tests. BBj Unit Test gives us just that – the ability to define a “setup” method that it calls exactly once before any of our unit tests run using the **@BeforeClass** annotation remark. We can add a method to **TestMathOperations** that will do exactly that. Our new method named **setup()** is shown in **Figure 4**.

```
rem @BeforeClass
method public void setup()
    #mathOperations! = new MathOperations()
methodend
```

Figure 4. Setup done exactly once before any of our unit tests run

The **@BeforeClass** is the right place for any initialization including preparing globals like STBL values, opening file channels, or any other preparation your tests require.

But what if we have setup code that needs to be executed before each unit test is run? Try **@Before**. Or cleanup code that needs to be executed after each unit test is run? Try **@After**. It turns out that a need for “setup” and “cleanup” code is pretty common, so BBj Unit Test offers you two different times to run setup and cleanup. **Figure 5** lists the recognized annotation remarks and their purposes, which includes both setup and cleanup options.

Annotation	Purpose
@BeforeClass	Run this method once before running any @Before or @Test methods (class setup)
@Before	Run this method immediately before running each @Test method (method setup)
@Test	Run this method as an actual unit test
@After	Run this method immediately after running each @Test method (method cleanup)
@AfterClass	Run this method once after running all @After and @Test methods (class cleanup)
@Ignore	Do not run this method as a unit test (it shows as ignored in the test results)

Figure 5. Annotation remarks for unit testing

How do I Determine Success or Failure in my Unit Test?

But how do we actually determine success or failure in each **@Test** method? Currently, BBj Unit Test offers a number of methods to determine this (see **Figure 6**).

Method	Explanation
Assert.Equals(BBjNumber A, BBjNumber B)	Successful if the number A equals the number B ; fails otherwise
Assert.Equals(BBjNumber A, BBjNumber B, BBjNumber Delta)	Successful if the number A equals the number B within +/- Delta ; fails otherwise
Assert.Equals(BBjString A\$, BBjString B\$)	Successful if the string A\$ equals the string B\$; fails otherwise
Assert.Equals(Object A!, Object B!)	Successful if the reference to object A! equals the reference to object B! ; fails otherwise
Assert.NotEquals(BBjNumber A, BBjNumber B)	Successful if the number A does not equal the number B ; fails otherwise
Assert.NotEquals(BBjNumber A, BBjNumber B, BBjNumber Delta)	Successful if the number A does not equal the number B within +/- Delta ; fails otherwise
Assert.NotEquals(BBjString A\$, BBjString B\$)	Successful if the string A\$ does not equal the string B\$; fails otherwise
Assert.IsNull(Object obj!)	Successful if the object obj! is null; fails otherwise
Assert.IsNotNull(Object obj!)	Successful if the object obj! is not null; fails otherwise
Assert.Fail()	Always fails
Assert.Expect(BBjNumber exceptionA, BBjNumber exceptionB)	Successful if the exception number exceptionA thrown equals the exception number exceptionB ; fails otherwise

Figure 6. Success and failure methods

With the exception of the **Expect()** method, all of the methods listed in **Figure 6** also offer a second method signature that is identical to the one shown but that takes a **BBjString errorMessage\$** as the first argument. The **errorMessage\$** allows you to specify a particular failure message to report if the invocation fails.

For the sake of simplicity, we will focus on the two simplest functions:

```
Assert.Equals(BBjNumber, BBjNumber),
and
Assert.Expect(BBjNumber, BBjNumber).
```

`Assert.Equals(#mathOperations!.add(8,8),16)` shows how to test the `MathOperations.add()` operation when we know the `BBjNumber` value we will get if the code is successful. In this case, we know that if we add 8 to 8 we will get 16, and our unit test will report success. If, however, `MathOperations.add()` returns any value other than 16, the unit test will fail.

`Assert.Expect(#mathOperations!.div(20,0),93)` shows how to test the `MathOperations.div()` operation under a condition when we expect it to throw a particular exception (93 in our example). In this case, we know what should happen if we try to divide any number by 0. The code should throw an exception 93, and our unit test should report success (because we got the expected exception). If, however, `MathOperations.div()` returns any value without throwing an exception 93 or throws any exception other than 93, the unit test will fail (because we expected an exception 93 but did not get it).

Putting All of the Pieces Together

Let's go ahead and define a full `@Test` method for each of `MathOperations`' methods: `addTest()`, `multTest()`, `subTest()`, and `divTest()`. Our unit test class now looks something like **Figure 7**.

You are not limited to testing only class operations like `mathOperations!.div()`. You can also invoke legacy functions like `CALL` as shown in **Figure 8**.

```
use ::MathOperations.bbj::MathOperations

class public TestMathOperations
  field private MathOperations mathOperations!

  rem @BeforeClass
  method public void setup()
    #mathOperations! = new MathOperations()
  methodend

  rem @Test
  method public void addTest()
    Assert.Equals(#mathOperations!.add(8,8),16)
    Assert.Equals(#mathOperations!.add(-9,8),-1)
    Assert.Equals(#mathOperations!.add(0,10),10)
    Assert.Equals(#mathOperations!.add(-9,-8),-17)
  methodend

  rem @Test
  method public void multTest()
    Assert.Equals(#mathOperations!.mult(8,8),64)
    Assert.Equals(#mathOperations!.mult(0,8),0)
    Assert.Equals(#mathOperations!.mult(-6,8),-48)
    Assert.Equals(#mathOperations!.mult(-6,-6),36)
  methodend

  rem @Test
  method public void subTest()
    Assert.Equals(#mathOperations!.sub(50,25),25)
    Assert.Equals(#mathOperations!.sub(5,25),-20)
    Assert.Equals(#mathOperations!.sub(-15,-15),0)
    Assert.Equals(#mathOperations!.sub(-15,0),-15)
  methodend

  rem @Test
  method public void divTest()
    Assert.Equals(#mathOperations!.div(50,2),25)
    Assert.Equals(#mathOperations!.div(20,-2),-10)
    Assert.Expect(#mathOperations!.div(20,0),93)
  methodend
classend
```

Figure 7. A complete example BBj Unit Test class, `TestMathOperations`

```
rem @Test
method public void callTest()
  CALL "myAddCall.bbj",5,7,x
  Assert.Equals(x,12)
methodend
```

Figure 8. An example of testing legacy code



Running Unit Tests

Now that we have defined our tests, it's time to run them. There are a number of ways to do this, but to keep it simple we'll run our tests from inside of the editor window. With the `TestMathOperations` class open in a code editor window, right-click anywhere in the code and select `Run As > BBJ Unit Test`. Our tests all run, and the BBJ Unit Test View appears as shown in **Figure 9**.

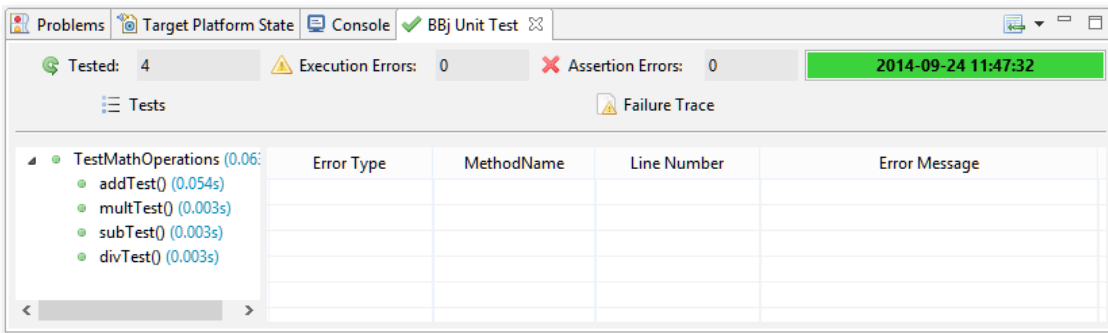


Figure 9. The results of successfully running `TestMathOperations`' unit tests

If we wrote all of the `TestMathOperations` unit test code correctly, and we wrote all of the `MathOperations` methods correctly, then BBJ Unit Test reports success and we get the nice green result display shown in **Figure 9**.

If, however, we made any coding errors in any of our unit test methods, we get a red result display with an 'Execution Error'. The view provides additional information to help us fix our mistake: the name of the offending unit test method, the line in that method that failed, and the matching error message as shown in **Figure 10**.

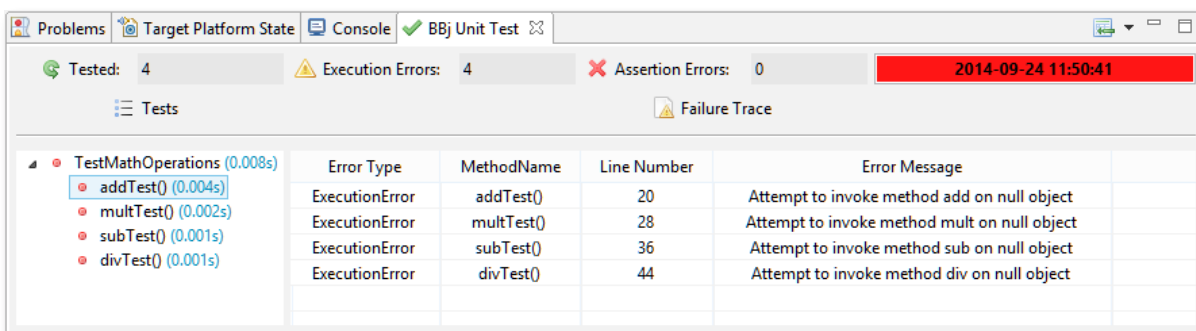


Figure 10. Execution errors in our unit test methods

If we wrote all of the `TestMathOperations` unit test code correctly but one `MathOperations` method works incorrectly, then BBJ Unit Test will report a failure for that test and we get a red result display. This error will show as an 'Assertion Error' indicating the name of the offending unit test, the assertion in that method that failed, and an error message to help us see what went wrong as shown in **Figure 11**.

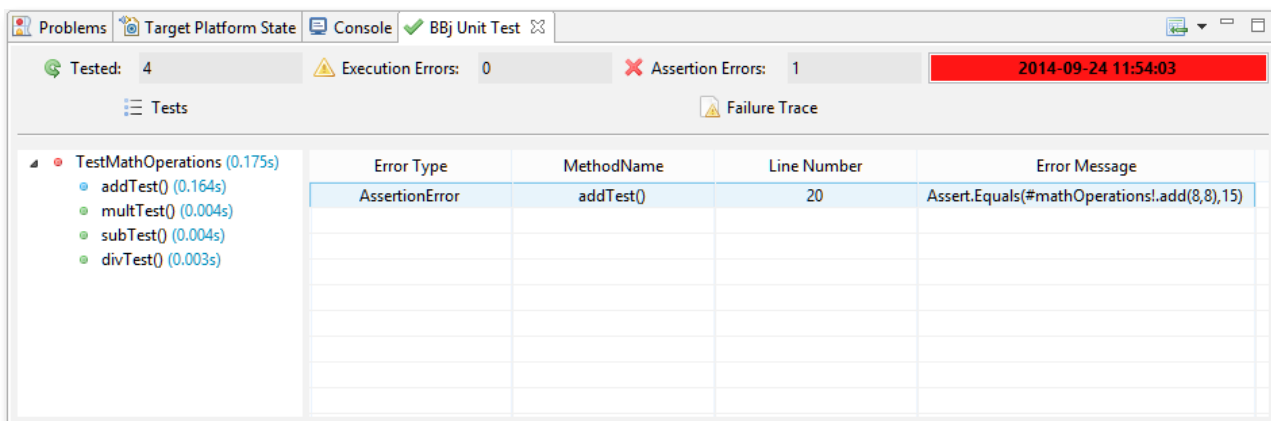


Figure 11. An assertion error in `addTest()`

If we made any mistakes, we now have to fix our problems and test again, and again ... and again. Remember, the goal of running our unit tests is to (eventually) see green, to have all of our unit tests run successfully. If we have written our unit tests to cover all of the necessary functionality, then success will mean just that – the code we tested (the MathOperations class) works the way we want it to. And we are ready to move on to our next coding task.

Summary

Unit testing is very valuable in determining whether or not a unit (such as a class or a function) works the way we want it to, to tell you when you are (finally) finished. One thing you may not have realized, though, is that the set of automated tests you just created can now be run again any time you modify your code or you prepare to release it. The industry calls this process [Regression Testing](#) and it can help you find any side effects or problems you may have inadvertently created before a release. The value of unit testing is not only in helping you know when you are finished developing, but it is also in helping you after that to know that everything is still working perfectly!

The BBj Unit Test Eclipse plug-in is free to use with the BDT Eclipse plug-ins and is built to be extensible, but it is a framework that is only in its infancy. It offers a number of useful methods to help you to unit test your programs, but there is still a great deal more functionality that we can add if you would find it helpful. So give it a try and send us your feedback on what you would like to see added next to BBj Unit Test. Help us to help you turn out better Business BASIC programs for your customers. ■



- Try out the BBj Unit Test plug-in today by installing [BBj Utilities](#)
- Download and run the [code samples](#)



The Anatomy of BBJ

As a new BBJ® user, you may find the landscape of participating processes and services confusing. Exactly what did I install on my server? What do all of these processes do anyway? In order to provide a firm grounding in the basics, let's look at a high-level overview of BBJ and its components.

At the Heart of BBJ

BBJServices is the core component of BBJ. It provides for central control and access to all data and BBJ programs, and provides local and remote file access, SQL processing, administration, user access, interpreter access and thin client support through these components:

- BBJ Data Server – Admin Server, Filesystem Server, SQL Engine, and the BBJ PRO/5 Data Server
- BBJ App Server – Interpreter Server, Terminal Server, Thin Client Server, Thin Client Proxy Server, Port Request Server, and the Jetty Web Server

See **Figure 1** for an overview of the BBJ components as they reside on the server "computer" with the remote client elsewhere. The BBJ Data Server components appear in blue; the BBJ App Server components are in green; client components are in yellow. Three core components appear in white – BBJServices, Enterprise Manager, and the License Manager.

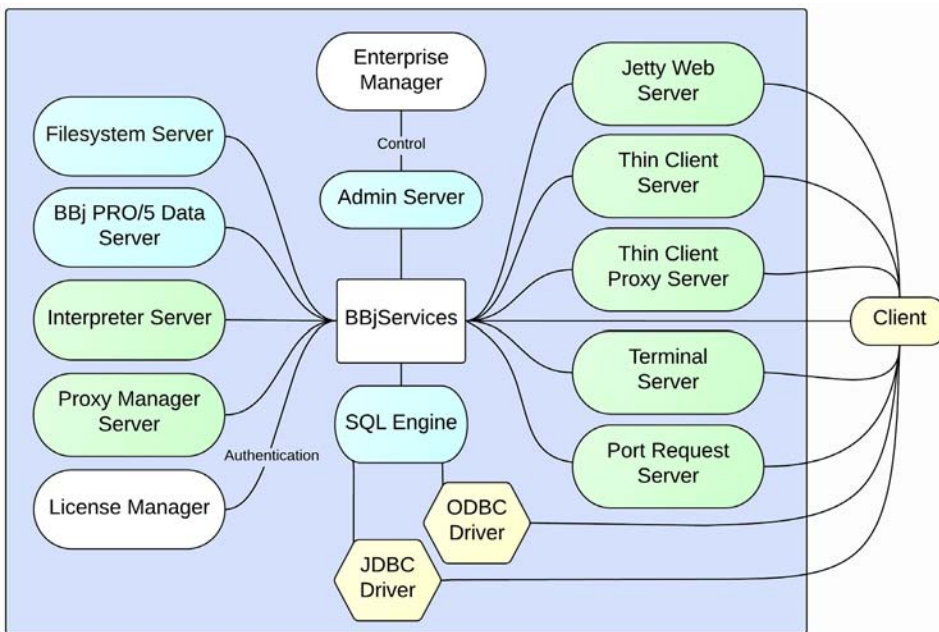
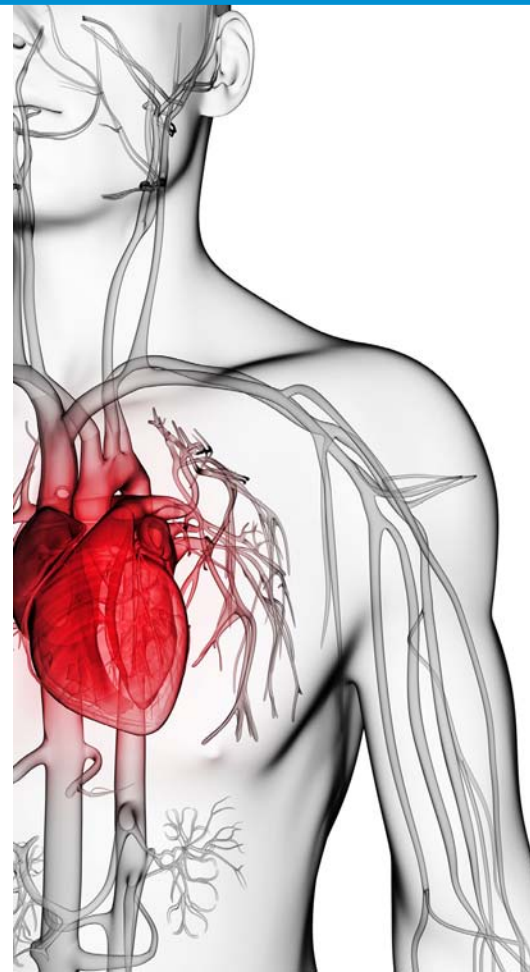


Figure 1. The BBJ components and their relationships



BBJServices

BBJServices offer the services of a data server (i.e. it listens on a given port for BBJ Data Server connection requests) and also offers the services of an app server (i.e. listens on a different port for BBJ App Server connection requests). When the App Server port receives a request, BBJServices starts an interpreter to process the request, but because the Interpreter and the Data Servers are running in the same process, they are able to communicate directly rather than across a socket. This provides a significant performance advantage over the configuration in which the Data Server and the App Server are running in separate processes.

Once BBJServices are running, users can access the services it offers in the same way that they would access the BBJ Data Server and BBJ App Server services if they had started either in separate processes. Although these services can be accessed in a manner that is 'pure Java,' the most common access is by typing BBJ at a command prompt.



Teresa Dominguez
Technical Support Analyst

Enterprise Manager

BBjServices is itself controlled or managed by an end user component, the Enterprise Manager (EM). The EM allows BBj administrators to configure and manage all the servers that make up your BBj enterprise. Administrators can configure the various servers, add users, define and maintain databases, configure database level permissions, setup and manage replication jobs, schedule tasks, create triggers, configure stored procedures, or any number of other tasks.

Two versions of the Enterprise Manager are available: a browser-based version and an Eclipse plug-in version. Installing BBj also installs a shortcut for the browser EM on each computer titled "Browser EM." You can also enter the URL <http://<localhost>:8888/bbjem/em> (replacing <localhost> with the hostname for your server) directly in a browser on any computer with network access to the BBj Server. The Eclipse EM plug-in is available to download and run within the Eclipse IDE development tool. Follow the installation and setting up instructions for the Eclipse version on the [Eclipse Plug-ins](#) web page.

BASIS License Manager (BLM)

The BASIS License Manager (BLM) is a licensing control process that checks for and serves up a valid license each time users launch or connect to a BASIS product. The BLM then monitors the license based on its expiration date.

BBj Data Server aka BASIS DBMS

Admin Server – The Admin Server is the server that does the following:

- Interacts with the BBj Enterprise Manager
- Allows for the configuration of BBjServices and network ports, and the logging of information
- Provides database configuration and user administration

Filesystem Server – The Filesystem Server handles access to all of the file types that BBj supports, both local and remote. It also manages data replication and change auditing.

SQL Engine – The SQL Engine provides server-side execution and processing of SQL statements. This allows for more efficient processing of multithreaded SQL statements than if the SQL Engine was part of the client-side ODBC driver. The SQL Engine uses an intelligent query optimizer that implements optimization strategies, including those for statements containing one or more ORs in WHERE clauses.

BBj PRO/5 Data Server – The BBj PRO/5 Data Server provides access to the BBj file system from Visual PRO/5® and PRO/5® clients replacing the PRO/5 Data Server®. Using the BBj PRO/5 Data Server users can now run their application with PRO/5, Visual PRO/5, and BBj simultaneously without data or licensing conflicts. This allows all clients to take advantage of BBj features such as directly accessing BBj ODBC data sources through the BBj SQL Engine, using stored procedures, triggers, database performance analysis, using extended SQL syntax, performing data replication and change auditing, and more.

BBj App Server

Interpreter Server – The BBj Interpreter Server starts interpreter sessions on the clients' behalf. The BBj Interpreter is the parser and interpreter of programs written in the BBj language. It supports multiple-line IF statements, non-line number programs, embedded Java code, reserved words, register/callbacks, dynamic limits and memory allocation, and ASCII programs.

Terminal Server – The Terminal Server provides a TermConsole BBj Interpreter session under Unix. It also provides IO interpreter sessions under Unix and Windows.

Thin Client Server – In thin client mode, the Thin Client Server runs the application specified by the client and keeps in communication with the thin client for user activity and additional processing. Speed is the primary goal in running all of the interpreter sessions in one JVM. It would be very inefficient to run a JVM for each session because the majority of the speed loss in Java occurs in the loading of the JVM.

Thin Client Proxy Server – The Thin Client Proxy Server (TCPS) provides access to Thin Client Servers from a local BBj executable. The TCPS works with the Port Request Server provide for the efficient management of JVM's used by the thin clients running on local clients.

Port Request Server – The Port Request Server interacts as a local server with the Proxy Manager Server to provide port information about the various servers to a BBj session when it starts.

Proxy Manager Server – The Proxy Manager Server maintains a list of currently active Thin Client Proxy Servers running on the local machine and interacts with the Port Request Server to insure a BBj Thin Client session connects correctly.

Jetty Web Server – The Jetty Web Server is a configurable web server. It provides a powerful tool to bring your BBj applications to web-based clients. Jetty-based applications can be BUI applications or JNLP applets such as Web Start applications. The Jetty Web Server also supports web services and comes with two built-in web services: the BASIS Update Service (BUS) and the demo Chile Company Ordering System (CCOS).

Clients

BBj Thin Clients connect to BBjServices and request that it run a BBj program on its behalf. BBj Thin Clients keep in communication with BBjServices and display the requested program's interface – either character user interface (CUI), graphical user interface (GUI), or browser user interface (BUI) – on the client machine. With BBj Thin Clients, BBj users can run any BBj program in thin client mode! You can also use BUI in a web browser on a variety of devices and machines. In this type of configuration, you can run a BBj application over the network without even having BBj installed on the user's machine or mobile device!

Database Drivers

You must use a driver to access the BASIS database management system (DBMS). This driver communicates with the BASIS database, giving it instructions to perform. BBj includes two drivers to access the SQL Engine; an ODBC (Open Database Connectivity) Driver and a JDBC (Java Database Connectivity) Driver.

BBj ODBC Driver

The BBj ODBC Driver provides an SQL interface to the BBj File System for third party applications running on Microsoft Windows. In previous releases of the ODBC Driver, the SQL process ran on the client. With the BBj ODBC Driver, the SQL engine is now part of BBjServices on the server, which makes the client-side driver very thin (small).

BBj JDBC Driver

The BBj JDBC Driver provides an SQL interface to any third party Java application running on any platform BBj supports.

Developer Tools and Application Building Blocks

Eclipse Plug-in – In order to develop BBj programs to run with BBjServices, use the Business BASIC Development Tools (BDT) plug-in for Eclipse. If you download and run BDT's CodeEditor, you will be able to create, edit, and debug BBj program code. Two new Eclipse plug-ins – AppBuilder and WindowBuilder (WB) – will be fully functional and available with the release of BBj 15.0. Leveraging the power of the Eclipse framework, these plug-ins offer a number of user-friendly utilities to make your development faster and easier.

- **CodeEditor** – provides an environment for BBj developers to create, edit, and debug their BBj code
- **WindowBuilder** – provides an easy-to-use rapid application development (RAD) GUI tool for BBj developers to do window design and layout (available in BBj 15.0)
- **AppBuilder** – provides a RAD GUI tool for BBj developers to create, edit, and manage event handlers for the UI created in WindowBuilder (available in BBj 15.0)

Barista – a GUI-only data dictionary-driven rapid application development environment and runtime engine, facilitates:

- New GUI application development
- Conversion of CUI applications to GUI
- Modernization of existing GUI applications

Refer to the Barista® [documentation](#) for a complete description of its features and components.

AddonSoftware – a set of enterprise resource planning (ERP) building blocks, is a full-featured and fully integrated business management solution powered by Barista and coded in BBj. It includes the following modules:

- **Accounting** – Accounts Payable, Accounts Receivable, and General Ledger
- **Distribution** – Inventory Control, Sales Order/Invoice Processing, Purchase Order Processing, and Sales Analysis
- **Manufacturing** – Bill of Materials and Shop Floor Control

Getting Started

To get started, download BBj from the BASIS [download web page](#) and choose only BBj or BBj plus the Barista Application Framework, with or without AddonSoftware®.

What's Next?

BBj continues to offer a number of flexible tools and options for the end user, and for the developer to design and manage a computing solution that fits the needs of their customers. This was a high-level overview of the landscape of participating processes and services that make up BBj to clearly explain what you installed on your server and what all of these processes do for you and your customers. ■



Makeover Your Images With BBXImage

Back in late 2008, when the BASIS engineers were in the midst of writing the LaunchDock Utility, they frequently found themselves dealing with images in application code. Oftentimes, the source image was *almost* what they needed, but didn't quite meet the application's criteria exactly. If only there was a quick and easy way to modify the source image programmatically, either at run-time or in the form of a batch image processing utility.

Necessity, as they say, is the mother of invention and as a result, the BBXImage Utility was born. As time went by, BASIS augmented the utility with more and more capabilities, solving multiple problems as the needs arose. As its functionality grew, other BASIS utilities and demos began to rely on its capabilities and so as of BBj® 14.0, BBXImage has achieved full-blown utility status and is now installed in the <BBjHome>/utils directory.

What Can it Do?

The BBXImage Utility offers a wide assortment of image-related services, so it's a challenge to describe it succinctly. Generally speaking, it offers several high-level features, including:

- Retrieving an image from a variety of sources
- Modifying the image in a number of different ways
- Saving the modified image out to a file or exporting it to another program

Retrieving an initial image is rudimentary, but the utility uses the BBXImageFactory class to create a BBXImage object from a variety of different sources. The image can be a BBjImage, a Java Image, an image file on the server, or even an image from a URL (such as one from the Web).

Most of the fun comes in when modifying the image, but once that is complete, you'll need to do something with the resultant image. The utility allows you to save it as a 32-bit PNG file with an alpha channel, a JPEG file with configurable compression, or retrieve a programmatic version of the image in a BBjImage, Java BufferedImage, Java Image, or Java ImageIcon format.



By Nick Decker
Engineering
Supervisor

Getting Information

Although it seems like basic functionality, we have found that often times our program needs to know the exact size of an image, but there's no easy way to get it. The utility solves this problem by providing getWidth() and getHeight() methods. In addition to being useful for the developer, the class uses this information extensively, such as when resizing the image while preserving the aspect ratio.

Image Modification

The real power of the BBXImage class is evident when it comes to modifying the image. The utility offers methods to accomplish several editing tasks, such as resizing, adding filters, rotating, flipping, cropping, rounding the corners of the image, adding a drop shadow, and more.

At BASIS, we use the BBXImage class internally for most of our screenshots to 'knock out' corners and add a border to the image. Many popular operating systems render windows with rounded corners, but image screenshot and capture utilities usually save out a rectangular image. The end result is a screenshot such as the zoomed-in version of a window shown in **Figure 1**, where the window does not have a border and the top left corner of the image shows a piece of the user's desktop behind the window's corner.



Figure 1. An enlarged top left corner of a typical screenshot with a section of the desktop visible



Figure 2. The same top left corner after using the BBXImage class to remove corners and add a border to the image

The BBXImage class easily solves this problem and fixes our screenshots, as we can take advantage of the roundCorners() and setBorder() methods. This removes the desktop background remnant in the corners and adds a matching border to the image. If desired, you can even add a drop shadow to make the image really stand out from its target background. The processed screenshot is shown below in **Figure 2**.

The utility has quickly transformed an average screenshot with distracting corner desktop remnants into a perfect specimen for documentation, presentation material, and marketing collateral. It also provides methods to round the top and bottom corners separately, so you can get the exact result you want regardless of how the original window rendered on the screen. Most of the manipulation methods, such as the roundCorners() and setBorder(), are flexible and take parameters to impact the extent of the image modification. That means that you can specify the border radius when rounding the corners, so it is possible to control how rounded the corners will be. The method also gives you the freedom to specify different sizes for the arc width and height, so the corners do not have to be circular. The screenshot in **Figure 3** shows six different examples where the border radius was set to ever-increasing sizes for the first four and the last two have different arc radii.



Figure 3. The effect of increasing the border radius on an image

As another example of the utility's flexibility when specifying a border for the image, you can stipulate the size, color, and opacity of the border. Adding a translucent border to an image adjusts the image's canvas size automatically, making room for the extra diameter. Saving the image as a PNG retains the opacity of the image as well as any drop shadows and borders that apply.

Other Modifications

To demonstrate just a few of the BBXImage's other manipulation capabilities, we'll start off with a simple image that we're all familiar with: the BBX® cup icon shown in **Figure 4**.



Figure 4. The original image before modification

With a few lines of code, we're able to modify the original image to suit our needs, as illustrated in **Figure 5**.



Figure 5. From left to right – the image is flipped horizontally, flipped vertically, rotated 15 degrees, has an added drop shadow, is changed to grey scale, set to 50% opacity, and darkened

The utility also offers a handy cropTransparent() method to return the smallest image possible by cropping out the transparent edges of the original. Best of all, because all of these modifications are done in a BBX program, you can transform batches of images programmatically. That makes it easy to automate the process of resizing, knocking out corners, and saving out multiple images into a target directory with a desired image format.

Resizing Images

We have seen it over and over again in movies – the detective takes a grainy traffic photo to the lab, enlarges it several times over, and ends up with a crystal clear image showing the perpetrator's license plate. Unfortunately, that's not possible in real life and just goes to show how far Hollywood is willing to stretch the truth to make a good story. In practice, resizing images is not easy and there are dozens of ways to accomplish the task. Most forms involve resampling the image, which relies on mathematical algorithms that interpolate and modify the data in the original image. When enlarging an image, many formulas exist to try to determine what the missing pixels would look like in a larger, higher resolution version of the same image. Interestingly, no one sizing method works best in all cases – each have their strengths and weaknesses. Some methods work best when enlarging images, and some are better at making images smaller. Some methods give passable results quickly, while others result in high quality images but take more time and computing power.

What BBXImage Offers for Resizing Images

The BBXImage Utility is flexible when it comes to resizing images. For starters, there are the `scale()` family of methods that attempt to 'do the right thing' when it comes to choosing a resampling algorithm. That means that the actual algorithm used may vary, particularly when comparing upsampling and downsampling an image. Using these methods, you can choose to scale your image by a percentage amount or provide a new absolute width and height. When you provide a scaling percentage, it retains the aspect ratio of the original image, but it can change if you provide your own width and height. Because retaining the aspect ratio is typically desired, BBXImage also offers other scaling methods where you provide the width or height and it automatically computes the new height/width, retaining the aspect ratio and scaling the image accordingly. **Figure 6** shows the result of scaling our BBj cup icon down while keeping the aspect ratio constant, as well as a version where we forced the width and height of the target image and overrode the aspect ratio.



Figure 6. A scaled version of the image with the original aspect ratio (left) and a distorted ratio (right)

Exerting Complete Control

In the few cases that you want or need control over the type of resampling to be used when scaling an image, the utility offers a `scaleWithHints()` method where you provide [RenderingHints](#). `RenderingHints` allow you to provide input into the choice of algorithms used by Java, which performs the rendering and image manipulation services. This means that you can tell the BBXImage class to resize your image using a specific technique, such as 'Nearest Neighbor', 'Bilinear', or 'Bicubic' interpolation. To get an idea of how the specific algorithms affect the final image, look at a saved screenshot of a Dashboard chart. We used BBXImage to shrink it down to 50% of its original width and height, so the resultant image contains one fourth of the information. To get a better idea of the resampling differences, we have magnified the results as shown in **Figure 7**.



Figure 7. A comparison between scaling with Nearest Neighbor (left) and Bicubic (right) interpolations

While the 'Nearest Neighbor' results in a blockier image with a lot of information lost, the 'Bicubic' version is smoother and retains more information. For example, both images contain the bars and numbers in the chart, but the 'Nearest Neighbor' version completely eliminated the chart's borders, axis, and grid lines. Although the 'Nearest Neighbor' algorithm is very quick, it often loses information as it only samples a single pixel adjacent to any that it processes. Upscaled images using this algorithm are usually blocky and those that are scaled down often completely eliminate some of the original pixels. In contrast, the 'Bicubic' algorithm is slower as it samples the colors of the nine pixels surrounding each one it analyzes, but the final result is a better approximation of the original.

Live Demo

While the BBXImage Utility's strength lies in its ability to manipulate images programmatically, you can take it for a test drive without writing any code. Just go to links.basis.com/bbximagedemo and run a BUI program that demonstrates a few of the more commonly-used capabilities. As you can see, the utility is easy enough to use that you can spruce up your adhoc images that you include in emails to important customers, as well. A screenshot of the BUI app appears in **Figure 8**.

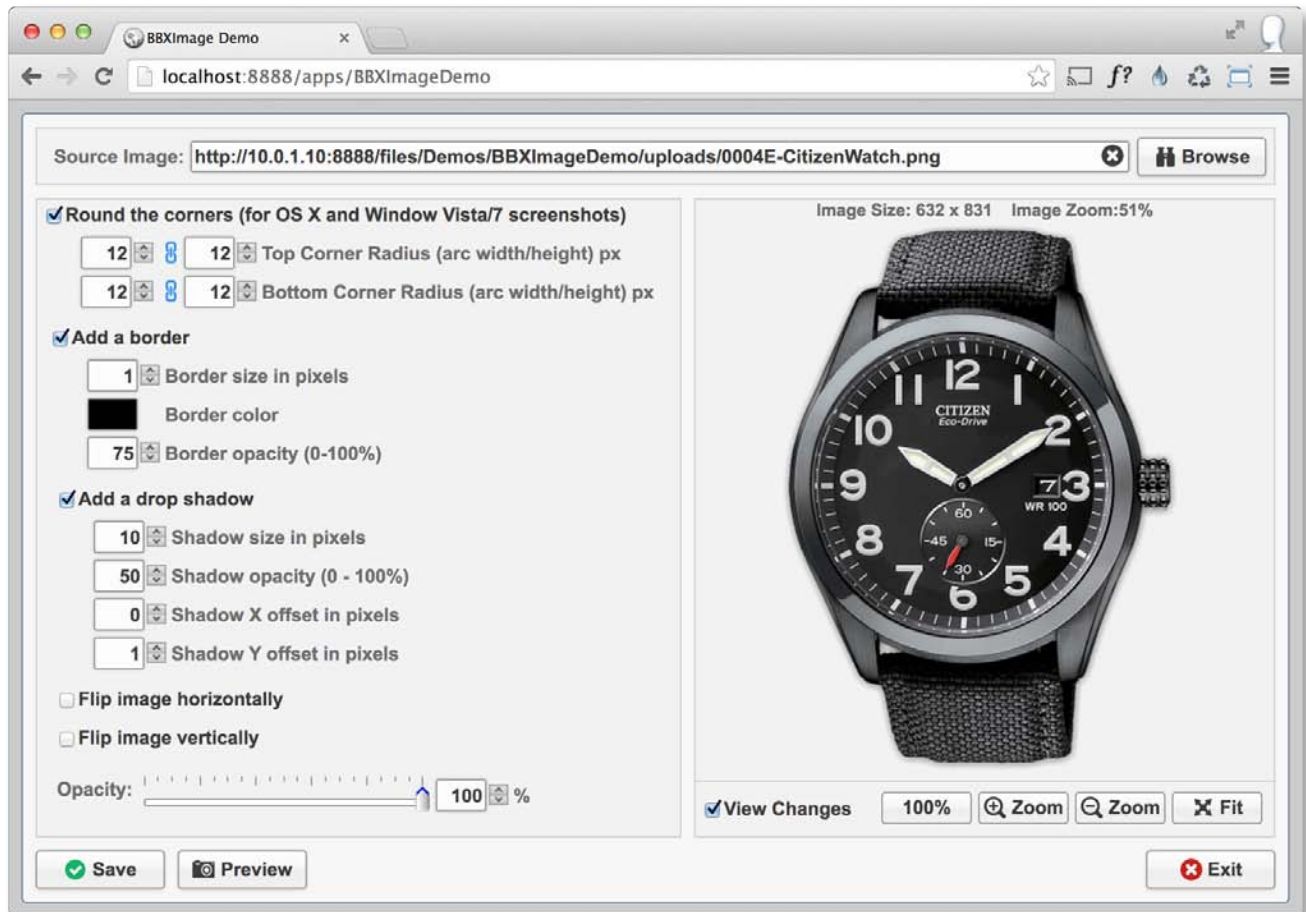


Figure 8. A BUI program demonstrates some of the BBXImage Utility's capabilities

Summary

The BBXImage Utility helps to fill a gap between BBj's graphical capabilities as offered by the BBjImageCtrl and Java's built-in image capabilities. Image processing can get deep, and typically the code required to accomplish seemingly simple tasks can be lengthy and rely on several low-level constructs. So instead of writing low-level Java code, why not use a custom BBj class that does all of the heavy lifting for you? The BBXImage Utility performs a number of common tasks so that your application images and screenshot material will look their best. Go ahead, satiate that starving artist that lives inside and make all your work look polished and professional! ■



- For more information, refer to
 - [BBXImage Overview](#) in the online Help
 - [BBXImage Class](#) JavaDocs
- Run the [BUI image demonstration](#) of the BBXImage Utility's capabilities

The Enterprise Manager Boldly Goes Forward

The Enterprise Manager (EM) has been a part of the BBJ® Product Suite since BASIS introduced BBJ in 2001. This powerful tool provides administrators with the ability to manage and configure one or more BBJ installations in a small office setting, or across the entire enterprise. As the features in BBJ (and technology in general) have grown over the years, so has the EM. This article explores just a few of the powerful features now found in the Eclipse/Browser EM.

Version 13.0 of BBJ introduced a completely new EM. Instead of continuing to maintain and add to what BASIS called the 'Java EM' or 'deprecated EM', we chose to take advantage of the latest desktop, browser, and mobile technologies and pass those on to our users. What does this latest technology give our valued customers?

- **An improved, modern interface** with the ability to open multiple tabs of information at one time making it easier to manage multiple parts of multiple servers and databases simultaneously.
- The option to use the **tightly integrated BDT EM plug-in** in the new Eclipse based IDE.
- Flexibility to **run the EM on virtually any platform** including desktops (Java and Eclipse required), web browsers (no Java or any additional plug-ins required), and even mobile devices such as smartphones and tablets.
- A product where all versions are built from a single codebase using the BBJ Admin API so that **each version stays up to date** with any new features, enhancements, and bug fixes; this provides the user with a common interface, regardless of whether they are running in the IDE, in a desktop browser, or from a tablet.

Superset

In BBJ version 14.0, the new EM is a superset of all the features and functionality found in the now deprecated Java EM. As a result, developers, and administrators should focus on using the powerful Eclipse/Browser EM rather than the older Java EM. Now, BBJ installations install a link to launch the browser-based version of the EM, making it quick and easy to get up and running with the latest technology.

Let's take a look at some of the new features in the Eclipse/Browser EM that are not present in the deprecated EM.



By Jeff Ash
Software Engineer

Multiple Tabs

Multiple tabs as shown in **Figure 1** give users the ability to have more than one item open for configuration at one time as well as information about multiple servers simultaneously. This new ability makes it much easier to compare configurations and go back and forth between panels without losing one's place. Furthermore, in the Eclipse plug-in version, users can rearrange the tabs for easy side-by-side comparisons or simply organize the panels in a more personally productive way.

Filters Open Files/Processes/SQL Connections

The new EM provides users with the ability to filter the list of open files, processes, and SQL connections to make it easier to locate the specific information required. Using the filtering feature, an administrator can locate all the processes for a particular program, or all open files for a particular user. This feature is especially helpful in large installations.

Incrementally Searches Log Files

Another very helpful feature is the ability to search for strings of text in server log files (see **Figure 2**). The search feature is incremental and has the option of only highlighting matching text or completely filtering out lines that do not have matching text.

Improves User Experience and Mobile Capability

BASIS significantly improved the user experience with plans for more improvements coming all the time. Currently, we have segmented the wizards into more easily digestible pages, categorized items in the navigation tree, given users multiple tabs to allow rearranging the workspace layout, simplified security management, and much more. Since the browser and Eclipse versions of

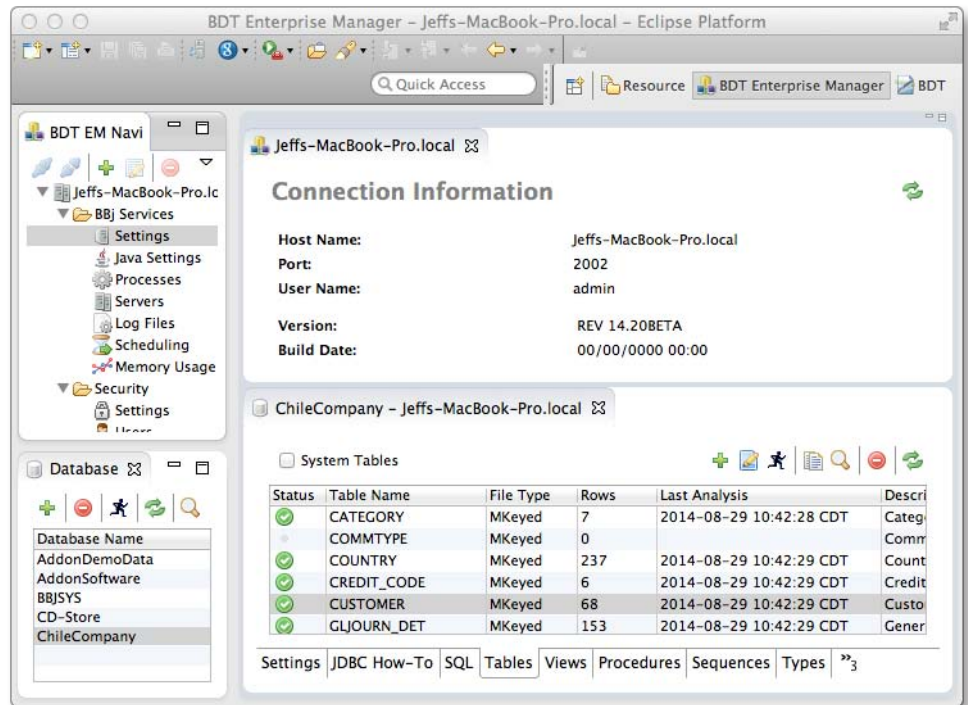


Figure 1. Multiple rearranged tabs in the Eclipse Enterprise Manager plug-in

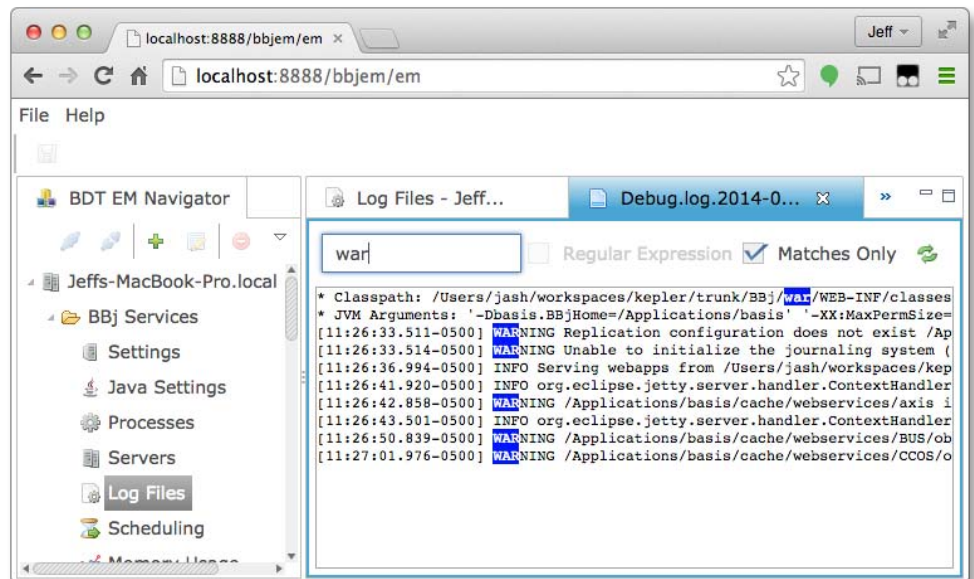


Figure 2. Incrementally search the contents of log files

the EM use the same codebase, the user experience is almost identical whether in Eclipse, a desktop browser, or a mobile device as shown in **Figure 3**.

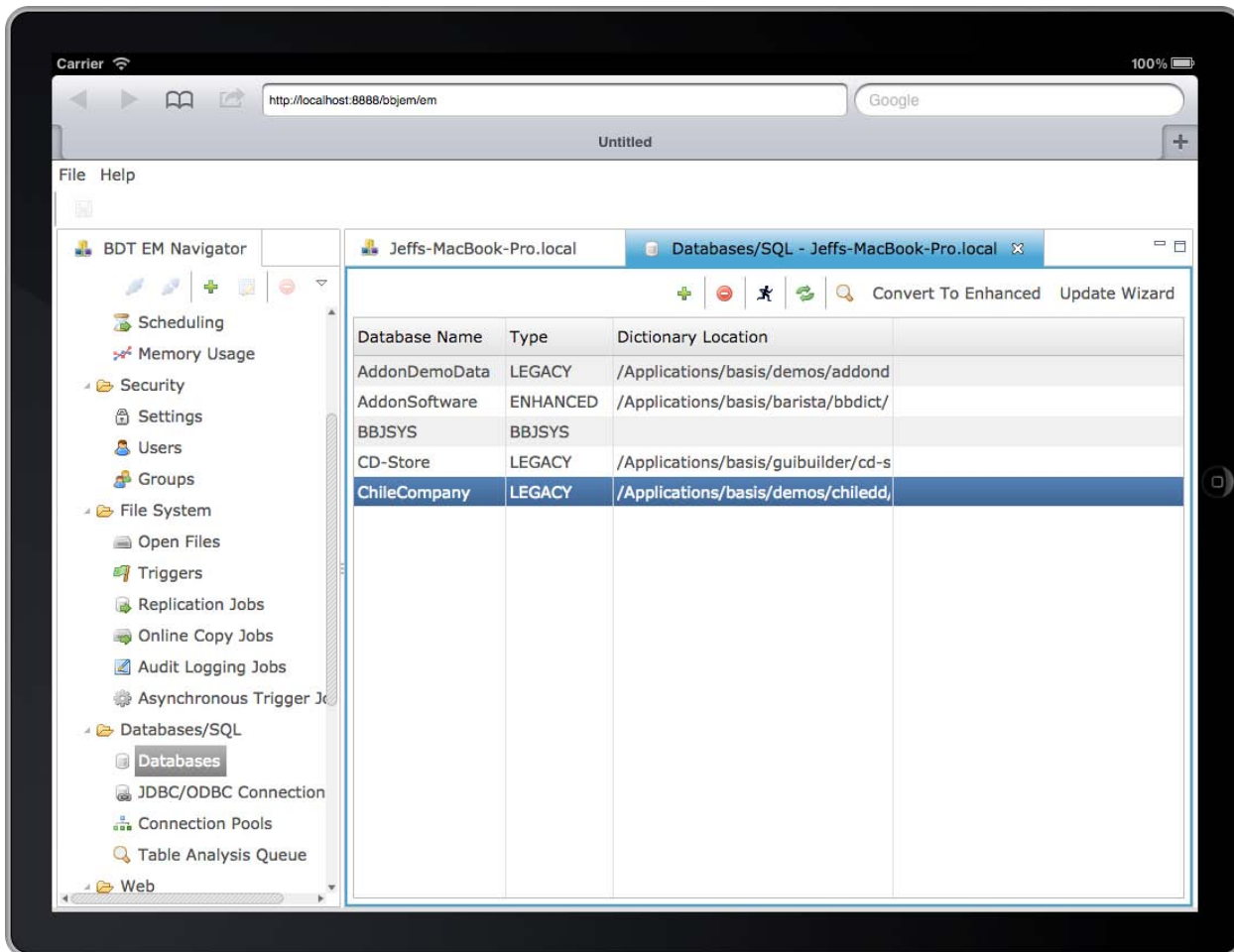
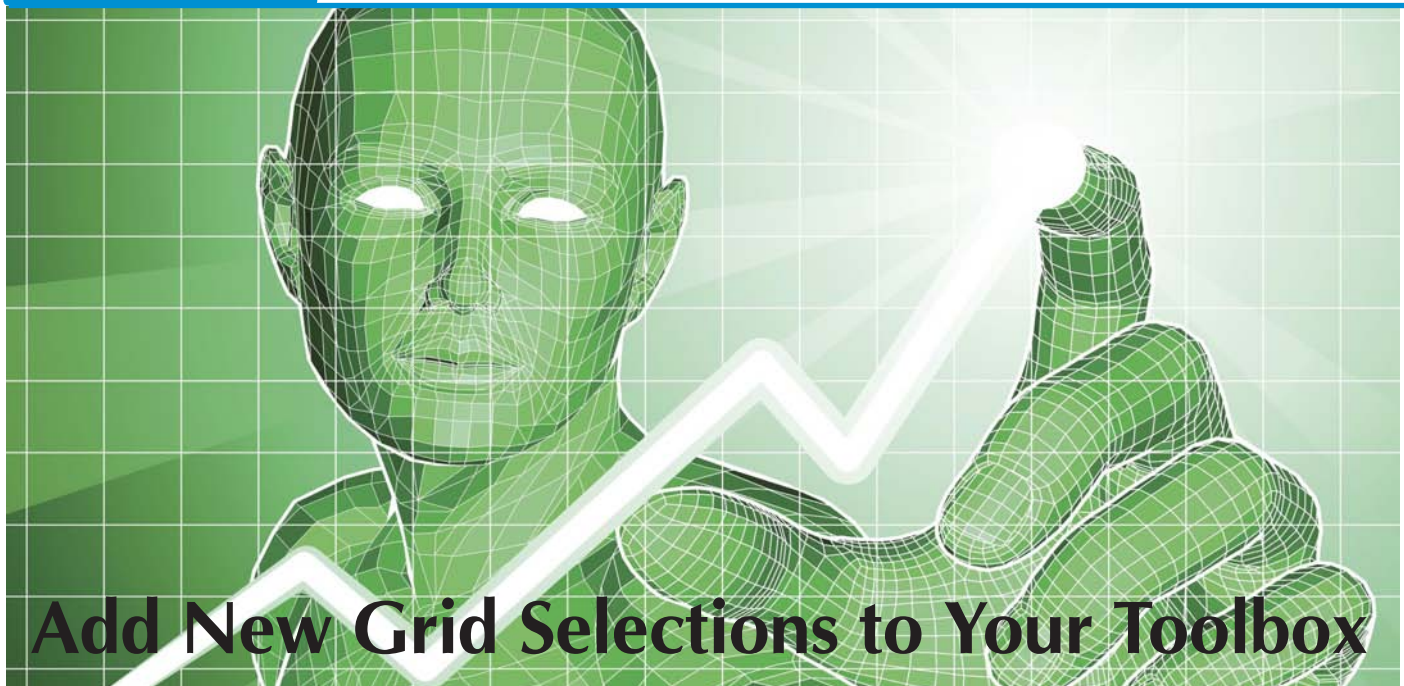


Figure 3. Use the Enterprise Manager on your favorite tablet or phone

Summary

The EM has grown from a simple configuration tool in BBj 1.0 to a powerful application for managing, configuring, monitoring, and testing all BBj installations across the enterprise. With the ability to run on a desktop, in a browser, and even on mobile devices, the flexibility of the latest evolution of the EM is sure to improve the productivity of all BBj developers, administrators and end users. Give the new Enterprise Manager a try, and “*Live long and prosper*” with the newest way to manage your enterprise. ■





Add New Grid Selections to Your Toolbox

Ideas can be seeds that explode onto the computers of an entire workforce but a lack of time, energy, or funding can starve an idea before it even gets a chance to germinate. In order to carve off time to nurture ideas when resources are tight, you constantly need to be on the lookout for ways to be more productive with the resources you already have. And getting productivity improvements to the tools in your toolbox, like a more fully featured grid control, just might help you find the time to feed your ideas so that they can take root and grow. This article looks more closely at how new grid enhancements will make you more productive in BBJ® 15.0.

BASIS Grid Controls

BBj programmers commonly use various Grid* controls in the role of data entry, data verification, graphical control layouts, and browser interface layouts. In a business application, the importance of organizing your data into rows and columns cannot be overstated. It can be paramount for users to view and access information contained in databases of any size, so much so that the Grid has entire functionality created for that exact purpose. However, as important as it is to access and view the returned results in a Grid, the information has little value unless a user can select data in the Grid.

*References to "Grid" in this article are intentionally not specific but refer to one or all of the BBJGrid controls – [BBjStandardGrid](#), [BBjDataAwareGrid](#), and [BBjDataBoundGrid](#) – because they use the same model for row, column, and cell selection.

Default Selection Model

BASIS made some massive improvements to the selection model used in the Grid to aid you in nurturing your ideas and turning them into reality beginning in BBJ 15.0. In the past, the Grid used Oracle's Default Selection Model (DSM) to work with the JTable, which is the base upon which we built our Grid. For years, the JTable with the DSM has met developers' needs, however, *The Times They Are A-Changin'* (Dylan, 1964). Although the DSM was effective at Grid selection, it had its limitations.

For example, Oracle based the DSM on the concept of rows and columns. Clicking in a cell with a mouse effectively split the selected data into two separate values, a row and a column. The selected rows and columns did not have any knowledge of one another and the Grid worked with them separately; however, a user could work with them in conjunction with each other under special circumstances.

In the modern world of grid manipulation, selecting by row and column causes problems for our user base. The most commonly reported problem is the inability to select and deselect multiple individual cells within the Grid. The DSM was only able to select a single cell because it recorded that cell as the intersection of the currently selected row with the currently selected column, deducing which cell the user actually selected.



By Aaron Wantuck
Software Engineer

The blue lines in **Figure 1** illustrate using a currently selected row and column to identify an individual selected cell, represented by a blue square. The selected cell is identified as (2,3) because cell references follow the format (row, col), where row and col are zero-based indexes, ignoring any headers.

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27

Figure 1. Selecting cell (2, 3) with zero-based indexes in the DSM

While selecting by row and column works well enough for a single cell or for multiple rows or columns, it does not work well for selecting multiple individual cells. For example, if you select cells (0, 5) and (2, 4), the DSM records that rows 0 and 2 are selected and columns 4 and 5 are selected. This produces a result that looks like **Figure 2**; you probably expected something like **Figure 3** instead.

Another common problem was that, at times, BUI selection would differ from GUI selection. This was a result of using the DSM for GUI applications while having completely different code duplicate the functionality of the DSM for BUI. It was difficult to maintain these two different code sets and still get the same selections under all conditions.

Enhanced Selection Model

To address these issues and provide an improved way to select items within the Grid, BASIS dropped both the DSM and the DSM-like BUI code, and combined them into a single shared code model, the “Enhanced Selection Model.” This means that multiple cell selection is now possible, and that future Grid code changes only need to be made once for both GUI and BUI.

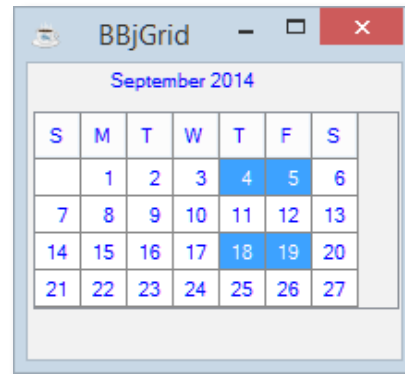
But what if you don’t want to change to the new model, what if the Grid selection you have used for years is “good enough”? You should not even notice the change. BASIS tested extensively to ensure that the default selection behavior in the Grid remains unchanged. We call this the legacy behavior, detailed further in the “Legacy Selection Model” section below. The Grid now offers you the new enhanced behavior, but only once you make a programmatic call to enable it by calling `setEnhancedSelectionModeEnabled(boolean)`. You can also call `isEnhancedSelectionModeEnabled()` at any time to find out whether your Grid is using the Enhanced Selection Model.

Improved Cell Selection

The Enhanced Selection Model does not use rows and columns, but records the individual cells selected either programmatically or through an input device such as the mouse. So what does this mean for Grid control users? Since each cell is recorded independently, you can now select multiple individual cells in a Grid. And you can use the standard [Ctrl]+click ([Command]+click on a Mac – hereafter just referenced as [Ctrl]) and [Shift]+click actions – to select or deselect one or more cells. For example, if you select cell (0, 5) and then hold down the [Ctrl] key while clicking cell (2, 4), you will now see the results in **Figure 3**.

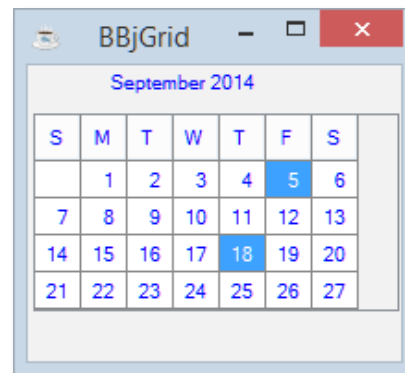
Also, you can now select a range of cells by holding down the [Shift] key while clicking a second cell. If after that you [Ctrl]+click individual selected cells, those cells will be deselected, resulting in never-before achievable selections such as those shown in **Figure 4**.

You do need to be aware of a few changes when using the new Enhanced Selection Model. The first is that the model brings with it the concept of the currently selected cell. Since you can now select multiple individual cells, which one should be returned when you call `getSelectedCell()`? BASIS now defines a *currently selected cell* as “the last cell selected, but only if it is still selected.” But what happens if you deselect the currently selected cell?



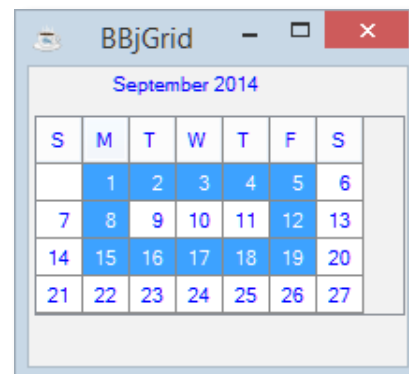
S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27

Figure 2. The intersection of rows 0 and 2 with columns 4 and 5



S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27

Figure 3. Selected cells (0, 5) and (2, 4)



S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27

Figure 4. An example of individual cell selection and deselection

Let's look at an example. If a user selects cell (1, 5), and then [Ctrl]+clicks cells (0, 1) and (3, 3) in that order, the currently selected cell would be (3, 3); `getSelectedCell()` would return (3, 3), and `getSelectedCells()` would return [(0, 1), (1, 5), (3, 3)]. The user will see the selections shown in **Figure 5**.

If the user then deselects cell (3, 3) with a [Ctrl]+click, then there is no longer a currently selected cell. Method `getSelectedCell()` would return (-1, -1) because the user removed the currently selected cell from the set of selected cells, but `getSelectedCells()` would still return [(0, 1), (1, 5)], and the user would see the selection shown in **Figure 6**.

Selection for Rows and Columns

The second change is that the Enhanced Selection Model no longer supports the `setSelectedRow()`, `setSelectedRows()`, `setSelectedColumn()`, and `setSelectedColumns()` methods. After all, you cannot select specific rows if there is no concept of a selected row; instead, you select cells. BASIS designed several new enhanced methods to allow you to set a selection – `setSelectedCells(Vector[intPair])` and `getSelectedCells()`. In BBj 15.0, these methods will allow you to select any combination of individual cells programmatically. The method `setSelectedCells()` takes a collection of integer pairs, each of which holds a zero-based row and column index representing an individual cell.

The `getSelectedRow()`, `getSelectedRows()`, `getSelectedColumn()`, and `getSelectedColumns()` methods are still available in the Enhanced Selection Model, but they work slightly differently than they did under the DSM. Now they return rows or columns that contain at least one cell that is selected.

To allow users to highlight an entire row, use the existing `setShouldHighlightSelectedRow()` method. So perhaps you are wondering, “If I use `setShouldHighlightSelectedRow()` to highlight full rows, how do I highlight full columns? A `setShouldHighlightSelectedColumns()` method doesn't exist.” Well, I am glad you asked. The Grid now offers a `setShouldHighlightSelectedColumn()` method that works with columns the same way as `setShouldHighlightSelectedRow()` works with rows. This new method is present in both the enhanced and legacy selection models, giving you full control over the Grid to highlight full rows and columns.

Legacy Selection Model

What happens if I already use methods that are unsupported in the Enhanced Selection Model? Well, the Legacy Selection Model mimics the behavior of the old Oracle DSM, and stores selection values in a row and column format just as it always has. You can use this legacy model to avoid changing your program, if you don't need to select individual cells, or if you simply prefer row and column selection. The Legacy Selection Model is still available for you and is in fact the current Grid default, so no change is required.

Summary

If you have an application that requires users to select individual cells in a Grid, then you need to use the new Enhanced Selection Model. The BASIS Grid allows you to have more control over how your users make their selections. It makes maintaining the code easier, and eliminates some of the shortfalls in the legacy Default Selection Model ... and it is optional. The Enhanced Selection Model is another example of BASIS providing you with better tools for developing your applications, while increasing your productivity, helping you turn your ideas into reality. ■

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27

Figure 5. Multiple cell selection with (3, 3) as the currently selected cell

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27

Figure 6. Multiple cell selection after [Ctrl]+click to deselect cell (3,3)



By Jerry Karasz
Software Architect

Building WindowBuilder

Once you decide to create a new BBJ® application with a graphical user interface (GUI), what is next? How do you get started? You probably already have a good idea of what you want the program to do and what you want your users to see. You may even know how you could create your GUI controls in BBJ code, but you are looking for an easier way to lay them out than manually creating the code. Wouldn't it be better if you could see how they look as you lay them out?

The BDT Eclipse development environment is now your best choice for GUI and BUI development. The BASIS NetBeans IDE is still a valid tool to use for the layout, but now you have an alternative under construction. Using the Business BASIC Development Tools (BDT) in Eclipse, you can download and run its CodeEditor Eclipse plug-in, and you will be able to download the AppBuilder and fully functional WindowBuilder (WB) plug-ins with BBJ 15.0. These plug-ins support Java 1.8, whereas the BASIS NetBeans IDE only supports up to Java 1.7. Eclipse offers a number of user-friendly utilities that will make your development faster and easier.

To implement a BBJ GUI application fully, you need to perform three tasks:

1. Lay out the graphical controls
2. Associate handlers with your controls' events
3. Write code to implement your business logic

This article focuses on the WB Eclipse plug-in as it exists today (as the alpha release that accompanied BBJ 14.x), which helps you lay out your graphical controls on one or more top level "windows" and "child windows" to match your design. For our purposes, we will assume that you have a BBJ project named **WindowBuilder1**, with an .arc file in it named **WindowBuilder1.arc** that is empty; if not, follow the directions at *Creating a WindowBuilder Project* (links.basis.com/creatingwinbuild) and create the BBJ project and .arc file. Delete all of the controls placed there when you created your .arc file, and start with just the 'Composite' in the drawing area as shown in **Figure 1**.

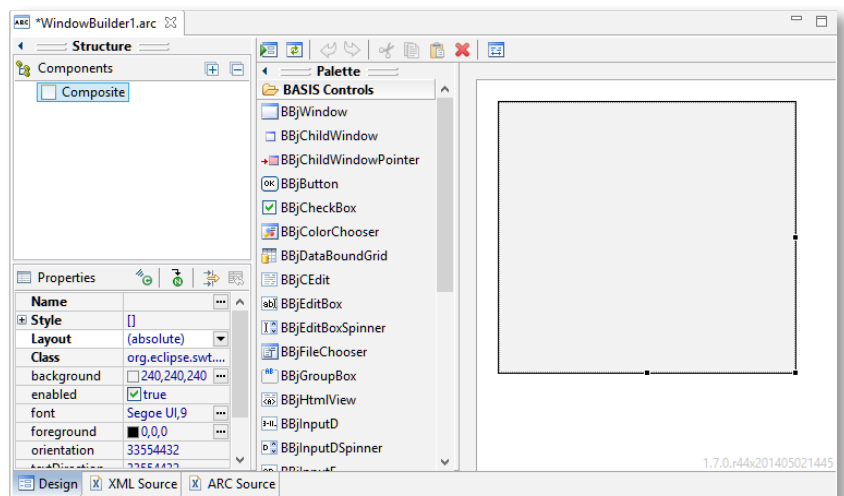


Figure 1. Starting with an empty .arc file

Features Today

Let's take a moment and look at the Eclipse layout. Because you asked to edit `WindowBuilder1.arc`, Eclipse opened an editor for you, the WB editor. This editor displays as one tab labeled with the `.arc` file name that in turn has its own three tabs – 'Design', 'XML Source', and 'ARC Source'. By default, the 'Design' tab displays but if you click on the 'XML Source' or 'ARC Source' tab, you can view the XML or ARC file representation of the WB controls you have placed. The XML is of no interest to you as it is just an intermediary format used by Eclipse. The ARC text is what WindowBuilder will actually write out to your `.arc` file when you save your changes.

For now, let's stick with the 'Design' tab and look at the three areas it offers: 'Structures' (made up of 'Components' and 'Properties' panes), 'Palette', and the unlabeled drawing area. The 'Palette' area shows all of the installed control palettes – **Figure 1** showed all of the default palettes disabled except for the 'BASIS Controls' palette. To change which palettes should display and how, use the Palette Manager (right-click any palette and choose [Palette Manager...]).

Your first task is to add a top-level window to the drawing area, so in the 'BASIS Controls' palette, click to select `BBjWindow`. Then click in the gray portion of the drawing area to place your `BBjWindow`. Notice that several things happened when you did this. First, a rectangle appeared in the drawing area that you selected and thus displays resize points (small black squares) that you can click and drag to resize it (see **Figure 2**). But more than that happened; there is now a strange entry selected in the 'Components' pane, `p1:BasisWindow` as illustrated in **Figure 3**.

The entry `p1:BasisWindow` is the XML element name for the `BBjWindow`. In fact, if you click on the 'XML Source' tab you will now see a `p1:BasisWindow` XML element has appeared like shown in **Figure 4**. Correspondingly, if you click on the 'ARC Source' tab, you will now see a `WINDOW` entry appear (see **Figure 5**).

Going back to the 'Design' tab, you may notice another change that took place. The 'Properties' pane now displays properties – in this case, for the selected item in the drawing area, the `BBjWindow`. If you expand the 'Bounds' property, you see the individual values: x, y, width, and height as **Figure 6** displays.

Your next task is to resize the `BBjWindow` to occupy most of the drawing area. You can do this by clicking and dragging the resize point in the lower-right corner of the `BBjWindow` in the drawing area, or you can click and edit the width and height properties directly in the 'Properties' pane.

It is now time to add a control in your design to your `BBjWindow`. To add any control, scroll in the Palette/BASIS Controls pane and click to select it, and then click to place it where you would like it on top of the `BBjWindow`. Continue to add controls, setting their properties as you go, until your window is complete. Finally, save your work by selecting `File > Save` from the main menu, by using the `[Ctrl]+S` shortcut, or clicking the [Save] toolbar button.

Now let's talk about some of the fun new features that Eclipse offers us for WB.

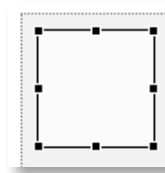


Figure 2. A `BBjWindow` selected in the drawing area

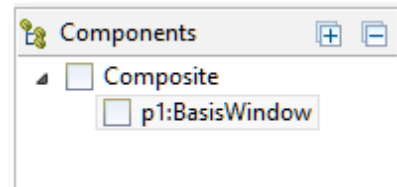


Figure 3. A `BBjWindow` in the 'Components' pane

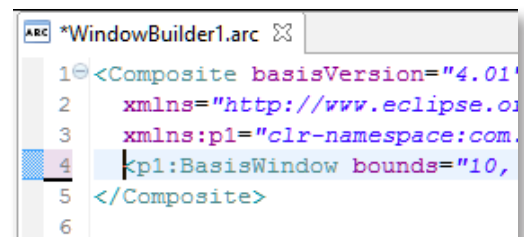


Figure 4. `BBjWindow` in the XML Source as `p1:BasisWindow`

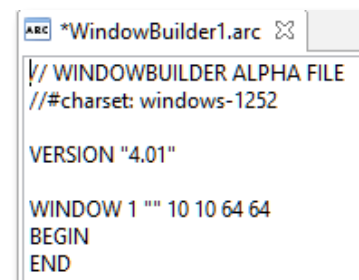


Figure 5. `BBjWindow` in the ARC Source as `WINDOW`

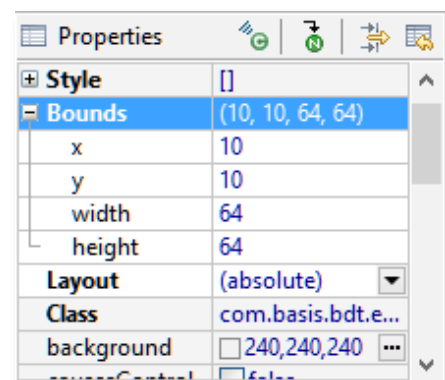


Figure 6. `BBjWindow` properties in the 'Properties' pane

Locate the button on the WB editor toolbar shown in **Figure 7** that has the hover text 'Quickly see/preview the window without compiling or running it'. Click this button and a small popup window appears showing you what your screen will look like when you use your new .arc file in a BBj program.



Figure 7. Toolbar button for previewing your BBjWindow

WB assists you with aligning your controls as you drag them onto the BBjWindow. If you drag a control to be near another control, WB will automatically show an alignment indicator (a line) connecting any side of the controls that is aligned (see **Figure 8** for an example when dragging a checkbox whose top aligns with the button). Notice also that the current X and Y coordinates of the control display as you drag it (112 x 10). This allows you to locate your control exactly where you want it. Remember, if you don't want to drag-and-drop your control, you can still select it and set the 'Bounds' values (x, y, width, and height) in the Properties pane to get the same effect.

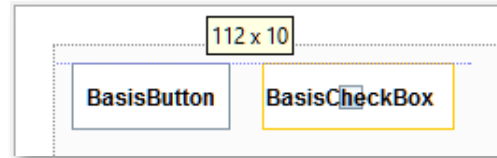


Figure 8. Alignment indicator while dragging a checkbox control

In addition, once you select more than one control, you have access to all of the standard alignment, movement, and spacing toolbar buttons shown in **Figure 9**, as well as the standard cut, copy, and paste functions. Possibly the best benefit of using BDT WB is that your control layout and editing can now be together with your code editing in Eclipse ... one powerful IDE to enhance your productivity that gives you a boost in writing code and in laying out your controls.



Figure 9. WB alignment toolbar buttons

Features Coming Soon

You may have noticed that WB is still under construction. BASIS is working hard to make sure that all of the appropriate functionality in Eclipse is available for WB users. This is a big task, and promises some exciting new enhancements for building GUIs in the near future.

Building Menus and Putting Controls on Tabs

We are currently developing the WYSIWYG model in WB that will allow you to easily build menus and add controls to tabs. The new drag-and-drop implementation for WB will make it simple and intuitive to build your menus, sub-menus, menu items, and popup menus by dragging-and-dropping items onto either the Structures pane or onto the Design tab. We are working similarly to enhance your experience when you place a control onto one of the tabs in a tab control – dragging-and-dropping a control and have it appear on the tab as it will at runtime.

Multiple Top-level Windows

We are working to provide a better way for WB to manage the multiple top-level windows you may have in one .arc file. In the BASIS NetBeans IDE, you could open a display for each top-level window, but interaction with them was limited by only being able to open one shared support window of each type: BBjGUI Palette, BBjGUI Inspector, and Properties. This meant that much of the editing could only occur on one top-level window at a time. WB will soon allow you to view and edit multiple BBjWindows at one time, where each display has its own 'Palette' and 'Properties' displays so that you can easily compare values and edit them.

.ARC File Properties

Another task is to extend the properties that are available for you to set and store in an .arc file. The format of an .arc file has been stable for many years, while the controls have continued to grow and offer more and more functionality. However, unless those properties can be stored in your .arc file, you cannot set them in WB and have them remain set when you run. Look for new properties to start appearing on many of the controls as we go forward – everything from setting the row gutter on a BBjGrid to setting the selected tab index on a BBjTabCtrl.

Summary

The WB Eclipse plug-in is BASIS' newest user-friendly tool to help you develop GUI and BUI applications quickly and easily. It will carry you into the future as Java 1.8 releases occur, and it offers a number of user-friendly utilities to make your development faster and easier. To make it even better, BASIS plans several enhancements for WB to leverage the power of the Eclipse development environment, not the least of which is the integration into Eclipse with the BDT CodeEditor. WindowBuilder is your best choice for graphical control layout for your GUI and BUI applications. ■



- Review [Creating a WindowBuilder Project](#) in the online Help or within the EclipseHelp
- Check out the [list of WB features](#) already implemented and those yet to be implemented



A New Day for AddonSoftware Partnerships



The AddonSoftware® ERP Partner Program recently introduced a no-membership-fee "Authorized Partner" tier that combines product discounts and free product training. This is an exceptional, low risk opportunity to help you expand your business!

The Opportunity

AddonSoftware partners enjoy generous margin opportunities, an exciting set of state-of-the-art business development tools, and the technical and training support they would expect from a long-standing industry leader of software development tools.



While the AddonSoftware Authorized Partner program tier has *no membership fees*, it extends significant product discounts; a higher discount on new/additional/upgrade licenses and a generous discount on software maintenance renewals. Suitably qualified Authorized Partners can quickly attain Premier or Elite status and begin growing their membership to the higher levels of the program.

The Product

AddonSoftware is an affordable full-featured and fully integrated business management solution. It offers scalable deployment options to fit your customers' needs and budget while providing the core enterprise resource planning features that currently support many businesses like those being run by your customers. Full integration eliminates the cost of multiple stand-alone applications and the resulting redundant data entry. With AddonSoftware, you select the licensing, define the number of users, and choose the functionality your customer needs.

Focus on Your Vertical

Do you have a vertical market that you want to put all your efforts into but your resources are divided maintaining standard ERP modules? Consider using the AddonSoftware ERP modules as your *building blocks* and focus all of your resources on maximizing the value of your vertical market solution to further differentiate you from the crowd. Using Barista®, a powerful and efficient data dictionary-based RAD customization tool, you can inherit built-in features and functions to get substantial new feature and function for your vertical with little or no effort.



By Paul D. Yeomans
Vertical Market Account
Manager

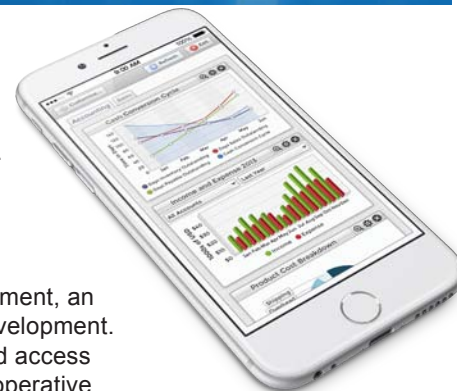
Supported by the E-Learning Center

Flatten the learning curve at the online e-Learning Center as it fits into your schedule, at the location of your choice, at your own pace. Read about it in depth at [On-Demand Enlightenment at the BASIS E-Learning Center](#).



The Cloud Option

Running AddonSoftware in the cloud makes deploying and upgrading your ERP software simpler and more affordable. Of course, you always have the choice of running AddonSoftware along with any BASIS solution on-premise. But, now you can run it online in a cloud environment – one solution, one technology set for both options. Cloud Services makes AddonSoftware even more affordable to use, own or rent, and deploy.



Commercial Open Source

Authorized Partners may optionally elect to participate in Cooperative Product Development, an innovative approach to earning product credits for ongoing AddonSoftware-directed development. Combining a traditional channel-based product delivery program with the openness and access to source code of an open source product development model, the AddonSoftware Cooperative Product Development Program delivers the best of both worlds, while retaining the retail value of the solution. Contributing expert product development meets a partner's own needs while helping to shape the overall future of AddonSoftware products. Your 'sweat equity' is rewarded with product margins of up to 100% on new product sales.

The Basics of AddonSoftware

- **Built-in features** – Includes a long list of standard utility features normally only provided by third party products or with additional development effort, thereby reducing costs.
- **Mature and yet contemporary** – AddonSoftware is the result of more than 30 years of continuous improvement with expanded features, increased functionality, and updated technology. For an example of these improvements, consider the graphical AddonSoftware Dashboard shown in **Figure 1** and featured in [AddonSoftware's Digital Dashboard Takes Off](#).

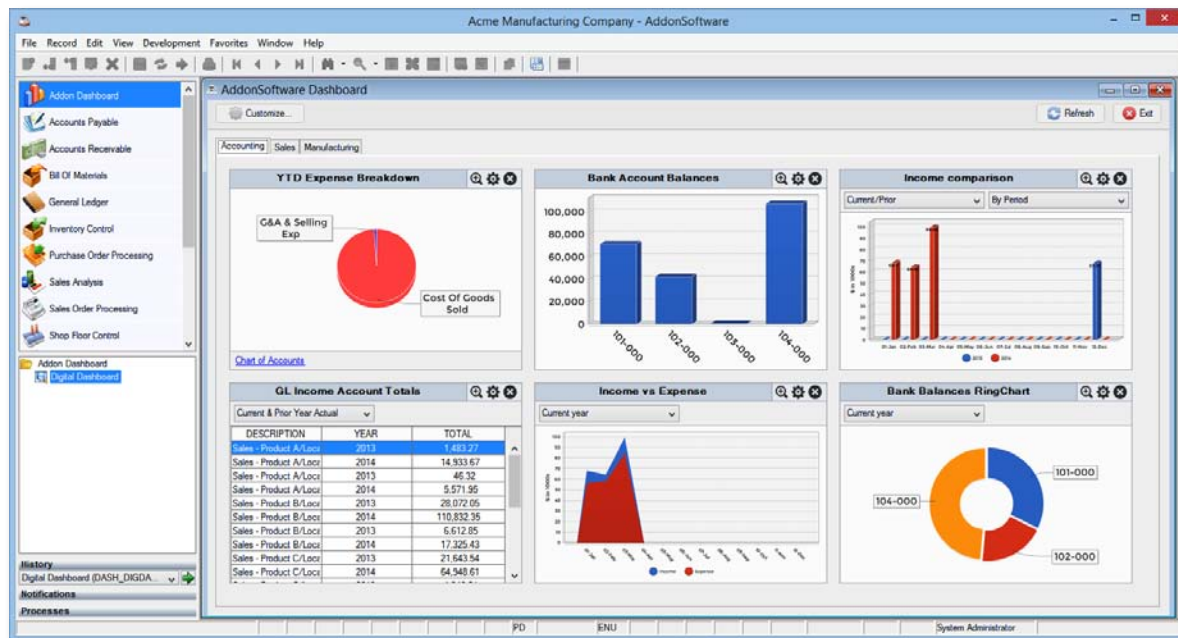
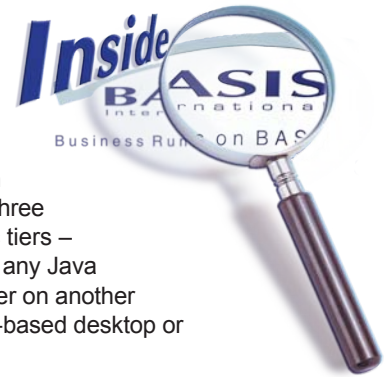


Figure 1. The digital AddonSoftware Dashboard

- **Easily customizable** – No one knows your business software needs better than you. Tailoring your business software to match your customers' workflow and industry practice is critical. To this end AddonSoftware includes Barista, the data dictionary-based RAD customization tool, at no additional cost.
- **Preserve your customizations** – Customized solutions often restrict upgrades to newer, more modern versions. AddonSoftware preserves years of customization investment across updates or even fresh installs of the base product.
- **Robust document output** – Users can preview, archive, print, email, fax, or convert to PDF or ASCII text format any generated report without additional costs. Reports may also be output to Google Docs for easy access in the cloud and real-time collaboration. Legacy report code is preserved in upgrade installations, and output accommodates modern printers, older line printers, and optional third party tools such as [UnForm](#).



- **Cross-platform freedom** – Choosing AddonSoftware does not dictate your operating system choices now or in the future. You can even choose to mix-and-match on any of the three deployment tiers – presentation, data, and logic tiers – where the application servers can each run on any Java supported operating system, the database server on another and the clients can be Macs, Windows, or web-based desktop or mobile devices without any Java.



Freedom to Grow

Becoming a partner is just the beginning. As you see, choice and flexibility are our core values. The partnership opens the door to the full suite of BASIS technology supporting, for instance, web and mobile application development.

Need a flexible e-business solution without all of the development work? The AddonStore shown in **Figure 2** is a multilevel platform for developing industry-specific e-business solutions. Best of all, you have the freedom of choice as to the accounting or ERP system the store interfaces with – AddonSoftware or another solution in your portfolio. Roll out your own e-business solution!

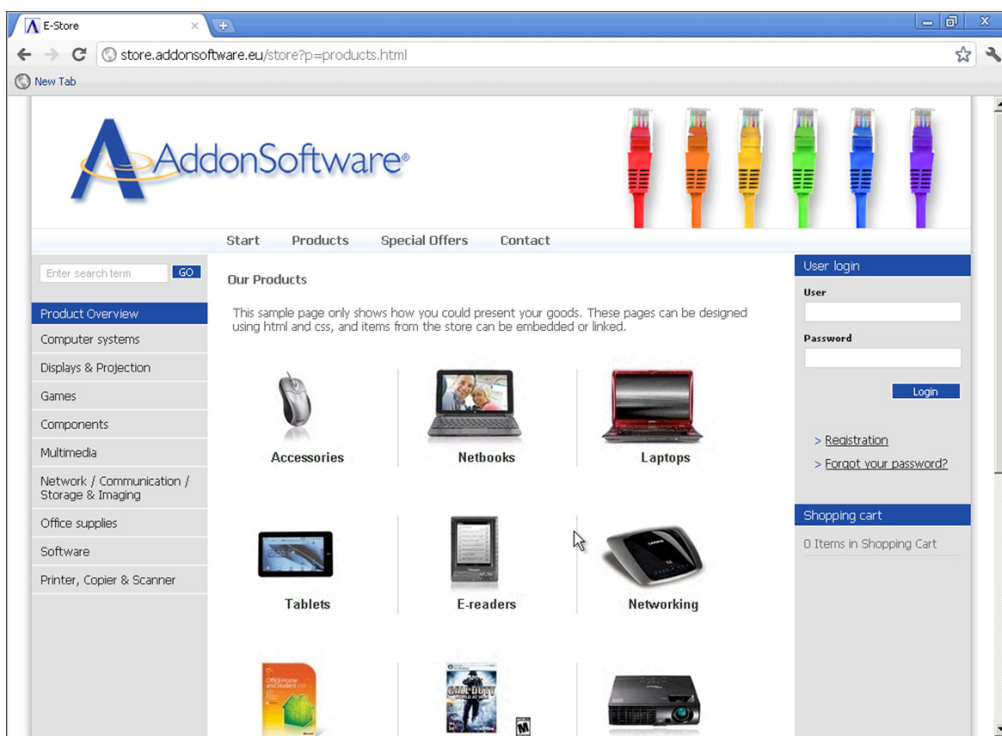
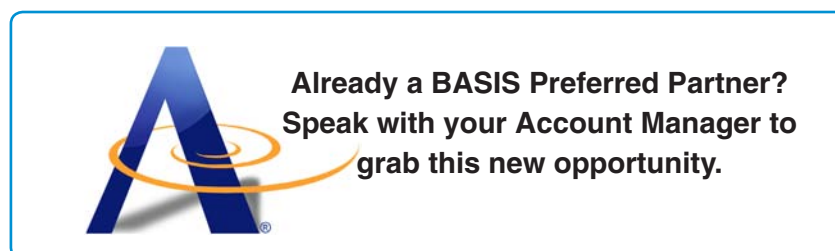
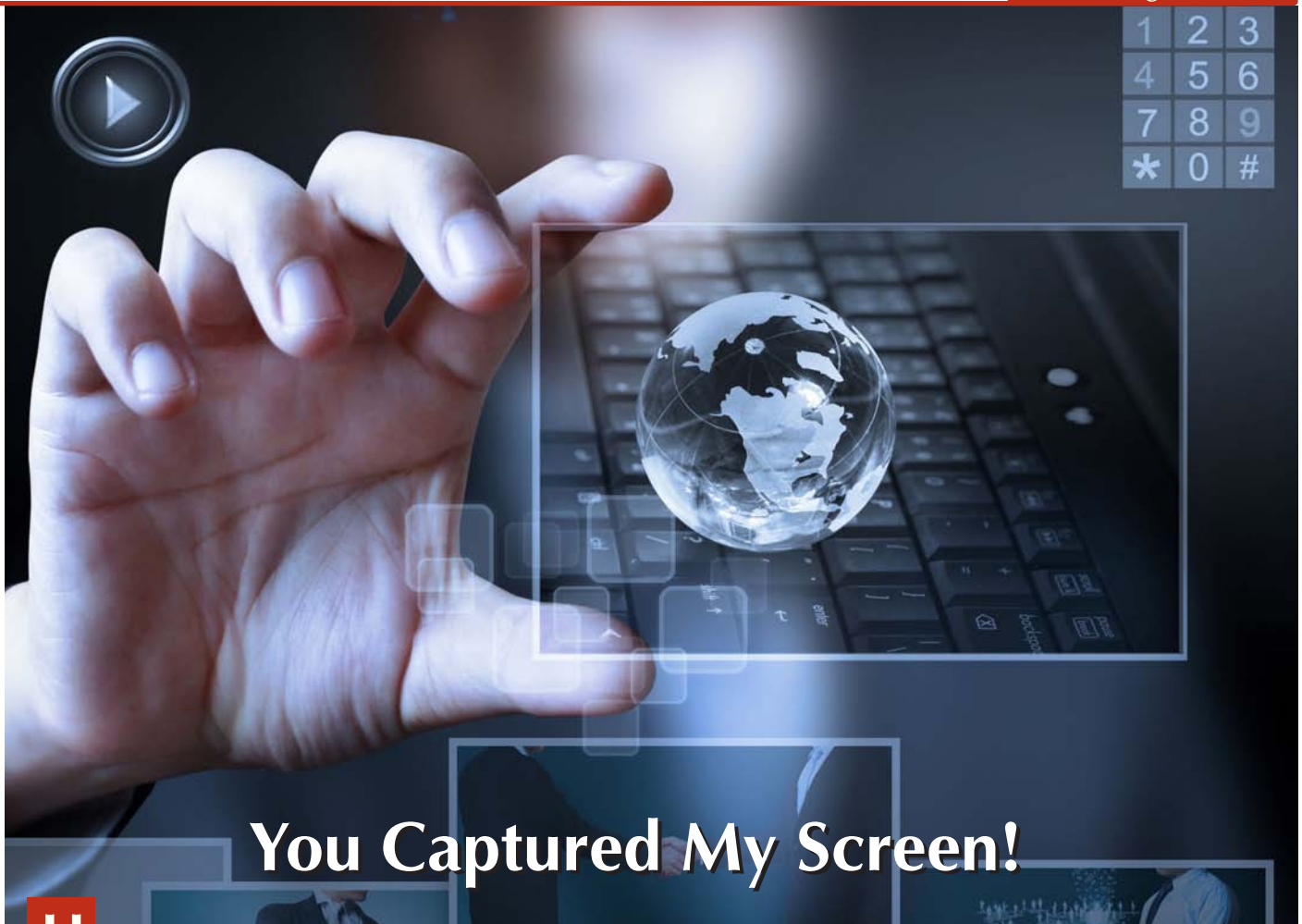


Figure 2. AddonStore, your e-business solution

Your opportunity is waiting. There is no downside to adding AddonSoftware to your portfolio. Contact us at info@addonsoftware.com.





You Captured My Screen!



How often have you thought, “If BBJ just had a function to capture an image of my screen...”?

If you are like many developers, you have thought this more than once over the years. And, after all, how hard could it be to just grab the screen contents and stuff it in a buffer, right? Unfortunately, it is not that easy, but with BBJ® 14.0 and higher, you can easily program a number of once-difficult tasks in only a few lines of code.

Through advancements in the BBJ language, you can now capture screens and all BBJ windows to BBJImage objects in both the graphical (GUI) and browser (BUI) user interfaces. BASIS has added a new [ScreenCapture](#) class to its arsenal of helpful utility classes in the file `<BBJ Install>/utils/screencapture.bbj`. The ScreenCapture class greatly simplifies your use of this UI-independent capturing functionality, giving you programmatic control over how to capture images.

In addition, we fine-tuned the existing [BBWindowUtils::centerWindow\(\)](#) method to take advantage of new [BBJTopLevelWindow](#) “getOuter” methods that deliver the outermost X and Y coordinates and dimensions of a window. The new versions of these methods take into account any window decorations such as a menu bar, window title, and frame. Until now, BBJ developers had to come up with their own algorithms to compute these values the best they could.

Enhancements to the BBJ Language

At the heart of the ScreenCapture class lie two capture-related methods that each return a [BBJImage](#) object.

1. [BBJSystemMetrics::getScreenImage\(\)](#) captures an image of the entire screen.
2. [BBJWindow::getWindowImage\(\)](#) captures an image of a top-level or child window.

Once you have a BBJImage object, you can manipulate it with code or stream it to a file.



By Ralph Lance
Software Developer

The `getOuter` methods added to the `BBjTopLevelWindow` class include the following:

- `BBjTopLevelWindow::getOuterX()` retrieves the outer window's X coordinate
- `BBjTopLevelWindow::getOuterY()` retrieves the outer window's Y coordinate
- `BBjTopLevelWindow::getOuterWidth()` retrieves the outer window's width dimension
- `BBjTopLevelWindow::getOuterHeight()` retrieves the outer window's height dimension

These methods allow you to position your windows more accurately, calculate the sizes and resize them, or perform a host of other previously complex actions where you need to know exactly what is where on the screen.

Screen Capturing Made Easy!

The `ScreenCapture` class installs with BBJ 14.0 and higher as a utility in `<BBj Install>/utils/screencapture.bbj`. It contains several static methods that you can invoke directly without having to first create (aka instantiate) an object instance of the class. `ScreenCapture` uses neither a client object instance of the popular `java.awt.Toolkit` for screen captures nor `java.awt.Robot` for window captures, so you can use it in both BUI and GUI programs.

Simply running `screencapture.bbj` brings up a demonstration window as seen in **Figure 1** that allows you to perform a screen, top level window, or child window capture.

By default, the demo program stores each capture image in a file in the 'Default temp directory' displayed near the top of the window. Marking the 'Show Save Dialog' checkbox gives you a 'File Save' dialog so you can override this default directory and store each image wherever you like. After saving your GUI capture to a file, it appears in the default image viewer on the client via a call to `BBjThinClient::browse()`. Captures made in BUI work identically, but their image file is made available to the web server via `BBUtils::copyFileToWebServer()`.

The demo code is at the bottom of the `screencapture.bbj` file, immediately following the classend statement for the `ScreenCapture` Class. The static methods in `ScreenCapture` allow you to use default values for pretty much everything. For example, you can use `ScreenCapture.capture([window!])` to use all of the default values, or you can set the image file path and name explicitly and specify whether the save file dialog should be presented using code like this:

```
ScreenCapture.capture([window!], temp_dir$ + "mycapture.png", 1)
```

The image file format used by `BBjImage::getBytes()` usually PNG, JPG, or GIF, is implied by the extension of the file name you provide.

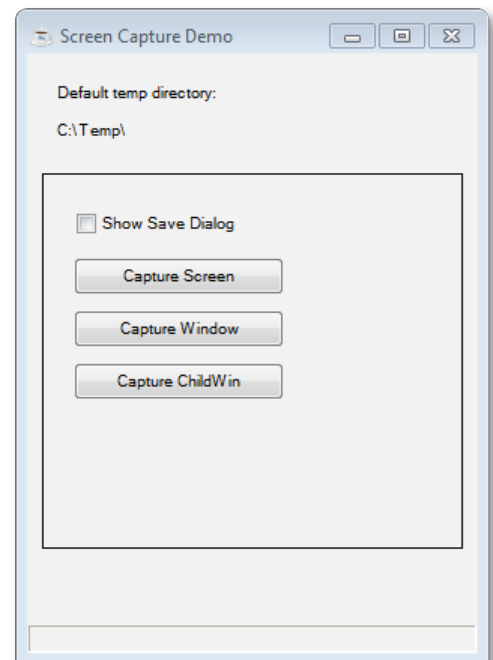


Figure 1. Screen Capture Demo

Summary

Due to the program control we now have, you can easily build screen and window captures into BBJ applications. One example would be in a common error handler routine that captures the state of the screen at the time of an error. Another use for this handy tool could be to "take a picture" of a top-level form or any part of a form contained on a child window programmatically while automatically creating rudimentary documentation and/or help files. The new `BBjTopLevelWindow` "getOuter" methods allow you to get the outermost coordinates and dimensions of a window without concerning yourself with window decorations.

These are just a couple of the most recent advances in the BBJ language that help BASIS create and enhance useful utilities for the BBJ community while giving you, the BBJ developer, more control with less effort. The possibilities for these new tools are limited only by your imagination! ■

 **Missed an Issue?**
Advantage
www.basis.com/advantage



By Jeff Ash
Software Engineer

Zero Deployment With JNLP

One great benefit web applications offer end users is the ability to simply visit a URL in their browser and run the application without needing any special installation or configuration. This functionality is possible because the browser provides the runtime environment for the client application. BASIS offers two powerful options for deploying a standard BBj® application in the browser. The extremely popular and cutting edge browser user interface (BUI) provides a quick and simple way to deploy a standard BBj application as a 100% browser-based application with no special installation or plug-in requirements for the user. However, some applications may be better suited for running on a standard desktop or they may have user interface requirements that are not supported by BUI such as custom Java components. Enter Java Web Start, a powerful part of the Java Runtime Environment automatically installed on most desktops around the world.

Java Web Start allows users to run applications directly from the Internet using a browser and without the sandbox restrictions placed on Java applets. The application does not run inside the browser, but rather, the browser downloads a JNLP (Java Network Launching Protocol) file that describes the application, configuration, and its required resources so that Web Start can properly run the application. When Web Start launches an application, it downloads the required resources described in the JNLP file and then runs the application. Since BBj's thin client is a Java application, BBj developers can make their applications available to their end users via Web Start to eliminate the need for any special installation on the client machines. Additionally, using Web Start makes it easier to upgrade to new versions of BBj because clients will receive an updated version of the thin client from the server automatically, once the server upgrade is complete.

Deploy an Application Using Web Start

Setting up an application for deployment using Web Start is quick and easy via the Enterprise Manager (EM) using the Eclipse plug-in version or the browser-based version. Initially, you only need to configure a few simple options to get up and running, however, the EM provides a very powerful and robust interface for managing the advanced aspects of the JNLP (including raw XML editing for those who require very specific changes).

1. Log in to the EM.
2. Expand 'Web' and 'JNLP Configuration' node in the 'BDT EM Navigator'.
3. Double-click 'Applications' to open the list of currently configured JNLP application configurations.
4. Select [+] to add a new application and open the 'JNLP Application Editor'.
5. At the top of the editor in 'Application Name', enter a name for the application without spaces, as shown in **Figure 1**.

NOTE: As you type the 'Application Name', the 'Launch URL' updates to reflect the changes. The 'Launch URL' is the URL to give to the end users to launch the Web Start version of the application.

6. In 'Application Description', type the program name and any additional program arguments in the 'Program and Arguments' field.

Additional options further down the dialog screen provide more configuration settings for the application as required.

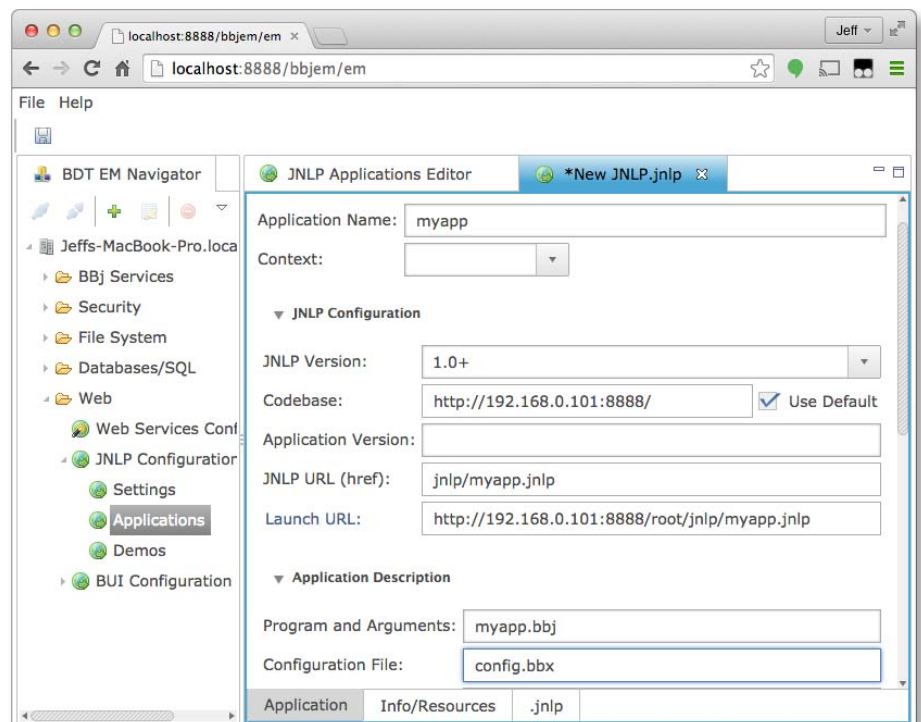


Figure 1. The top portion of the 'JNLP Application Editor'

Note the three tabs at the bottom of the editor – ‘Application’, ‘Info/Resources’, and ‘.jnlp’. Because of the many configuration options available for JNLP deployments, the editor splits the information into multiple tabs for better organization. The ‘Info/Resources’ tab provides the ability to include additional resources such as icons, additional JAR files, and even native libraries. Further, administrators can include different resources such as platform-specific JARs or native libraries based on the client’s operating system. **Figure 2** shows an example of two such JAR files included by default in each JNLP application. Applications require one JAR for Windows (**webstart2166.jar**) and an alternate JAR for Mac (**webstart2120.jar**).

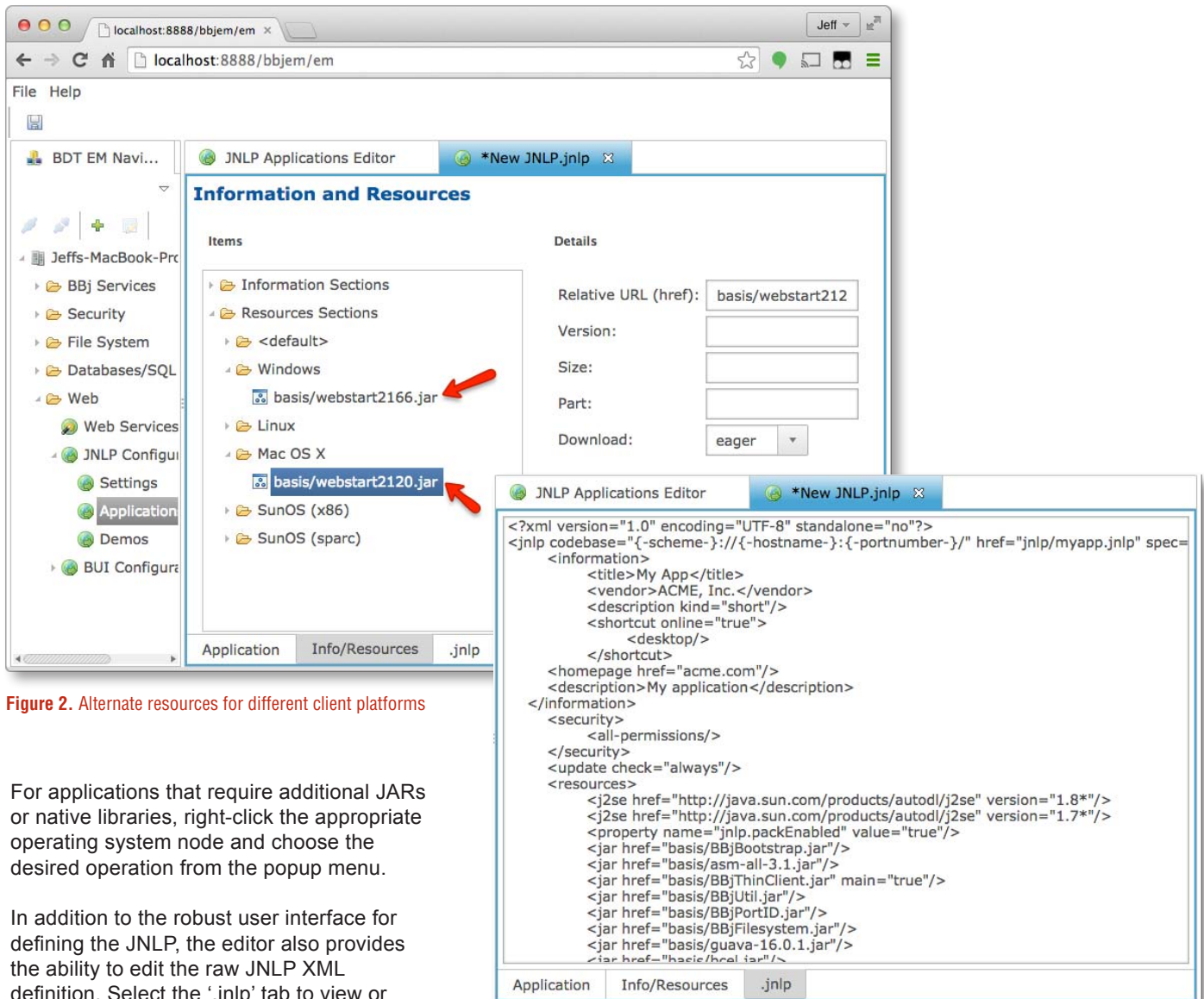


Figure 2. Alternate resources for different client platforms

For applications that require additional JARs or native libraries, right-click the appropriate operating system node and choose the desired operation from the popup menu.

In addition to the robust user interface for defining the JNLP, the editor also provides the ability to edit the raw JNLP XML definition. Select the ‘.jnlp’ tab to view or edit the XML definition (see **Figure 3**).

Figure 3. XML definition for the JNLP file

Summary

Java Web Start provides a number of powerful features that make deploying BBJ applications throughout the enterprise a snap. There is no need to install a BBJ thin client on each machine manually when Web Start can handle this automatically. Instead, using the EM, configure those enterprise applications to run from Web Start quickly and safely. Save time managing deployment, installation, and updating applications and put that time to better use making that already great application even better! ■



- For more information regarding the use of Java Web Start, refer to
 - [Configuring Web Start During a BBJ Installation](#)
 - [Running BBJ Thin Client with Java Web Start](#)
- Read these related Advantage articles
 - [Enterprise Manager Admin Only “Safe” Mode](#)
 - [Don’t Put All of Your Jetty Eggs in One Context](#)



Kurt Williams
Software Developer

Painless Payables - Anywhere!

Until now, processing invoices for payment in AddonSoftware® was a bit time consuming and prone to errors, not to mention that it required onsite approval and/or signature. However, the release of AddonSoftware 14.0 changed all that. It's now easy, efficient, and mobile!

The Payment Authorization feature in the Accounts Payable (AP) module now greatly improves record keeping, streamlines workflow, and saves tremendous time. The AP clerk handles paper invoices only once; they never leave the clerk's desk and are destroyed when the run is complete. The system manages the flow of work and executives can perform their tasks from anywhere, whether they are in their office or traveling on the road. Errors are minimal and time is saved!

This article looks at the AP process flow then and now, and offers some guidelines to using this new feature.

The Old “Painful” Way

The standard flow looked something like this:

1. An AP clerk received invoices from vendors, recorded on the face of the paper invoice which General Ledger account(s) to charge, organized the paper invoices into some kind of order, and then delivered them to an AP supervisor for payment approval.
2. The AP supervisor reviewed the paper invoices and signed or initialed them as approved for payment and returned them to the clerk.
3. The clerk
 - a. Entered the invoice data.
 - b. Printed the checks.
 - c. Returned the checks to the responsible executives for signing along with the paper invoices in case they had questions.
4. When the executive completed signing all the checks, the executive returned them with the paper invoices to the clerk.
5. The clerk mailed the checks to the vendors and filed the paper invoices in a filing cabinet.

“Painless” Invoice Entry and Review

In the new Payment Authorization feature, the simplified entry/review flow looks like this:

1. The AP clerk
 - a. Receives the invoices from the vendors and immediately enters the data, scans the paper invoice, and attaches the image to the invoice record in the system.
 - b. Notifies the reviewer, who is normally the AP supervisor, that invoice entry is complete and ready for review.
2. The reviewer, using the ‘Payment Selection Entry’ form
 - a. Looks at the invoice images online.
 - b. Records approval of the invoices as “ready to progress through the process,” all without touching any paper.
3. The approvers/check signers receive an email notifying them that there are invoices awaiting their approval.

“Painless” Invoice Selection and Approval

The new process of selecting and approving invoices for payment looks like this:

1. The approver/check signer, using the ‘Payment Selection Entry’ form
 - a. Reviews the invoice images online.
 - b. Records approval for each payment.
2. The AP Clerk
 - a. Verifies that all checks have the required approval or approvals as detailed in the Two-Signature Requirement section below, using the ‘Payment Selection Entry’ form.
 - b. Prints the checks on which the system applies the signature image of the approving executive.
 - c. Mails the checks to the vendors and shreds the original paper invoices.

Two-Signature Requirement

If any check exceeds the limit set by the business that requires two signatures, the AP module will require a second approval. The process looks like this:

1. The first check signer finishes and closes the ‘Payment Selection Entry’ form.
2. Check signers/reviewers receive an email notification detailing the status of the approval process.
3. A second check signer then logs on and reviews the invoices that need additional approvals.
4. The group receives a notification email with the approval status of the invoices.

Payment Authorization Setup

The Payment Authorization feature is configured via the ‘Account Payable Parameters’ form as shown in **Figure 1**.

Figure 1. ‘Payment Authorization’ tab in the AP Parameters form

This form controls whether to use and how to use the Payment Authorization feature. Here, you can specify whether to send notification emails, and whether two signatures are required and at what payment threshold the second signature is needed. It also controls where to store the invoice images, which can include Google Drive and the Barista Document Archive system. The system can require that each invoice has an associated stored image. If so, the system blocks the invoice data entry update if an invoice is missing a stored image. You can also set the background colors to visually identify the approval status of an invoice on the ‘Payment Selection Entry’ form.

Payment Authorization Approver & Signer Form

The new 'Payment Authorization Approver & Signer' form is where you set up the reviewer and approvers/check signers as shown in **Figure 2**. This form controls which users are the AP clerks, which are the reviewers providing preliminary approval, and who are the check signers, as well as storing the location of the signature image file. You can limit the check signer's approval authority by specifying a maximum authorization amount.

Figure 2. Payment Authorization Approver & Signer setup form

Payment Selection Entry

The reviewer and approvers/check signers do all their work in the 'Payment Selection Entry' form shown in **Figure 3**.

Figure 3. Payment Selection Entry form

The user can select individual grid rows (invoices) by clicking on a single row, select multiple grid rows by using the [Ctrl]/[Command] or [Shift] keys in combination with a mouse click, or select all rows by using the [Select All] button. Once users have selected a row or rows, they can click [View Images] to render the invoice images for the selected rows in the browser for review. Clicking [Approve Invoice] records their approval of the selected invoices for payment.

In **Figure 3**, you can also see the background color-coding of the invoice approval status. The first four invoices have all the approvals they need so they have a white background and the check box to the left is marked. The last three invoices have one approval, but require a second approval since they will result in checks greater than \$1,000. The background in our example is lavender and the check box is not checked. The [Clear All] button clears any selections that might have been made in error. When all selections are completed, the user clicks on the process button – a green arrow – thereby applying the approvals and exiting the form.

Figure 4 illustrates what a notification email looks like during the authorization process. Notice the upper grid in the email shows invoices with one approval, but require two approvals because the resulting check would be over the company-defined limit of \$1,000. Also notice the four invoices in the lower grid which have been completely approved and are ready for payment.

Approver JHANCOCK has exited the Payment Selection Form.

The status of Accounts Payable Invoices is as follows:

These invoices have been reviewed and have one approval, but require another approval:

0725-01	002439	Tires, Tubes and Accessories	1,010.00
0725-02	002494	Athletic Apparel	550.00
0725-03	002494	Athletic Apparel	600.00

These invoices have been fully approved and are ready for payment:

0701-01	000001	Sprocket Supply Co.	150.00
0701-02	000001	Sprocket Supply Co.	100.00
0701-03	000001	Sprocket Supply Co.	75.00
0701-04	000001	Sprocket Supply Co.	50.00

[Launch Barista Application Framework in browser](#)

Figure 4. Email notification of approval status

Check Printing

Once all the approvals are completed, check printing can proceed. The system applies a signature image to the check based on which signer(s) approved the invoice for payment. **Figure 5** illustrates a signed check.

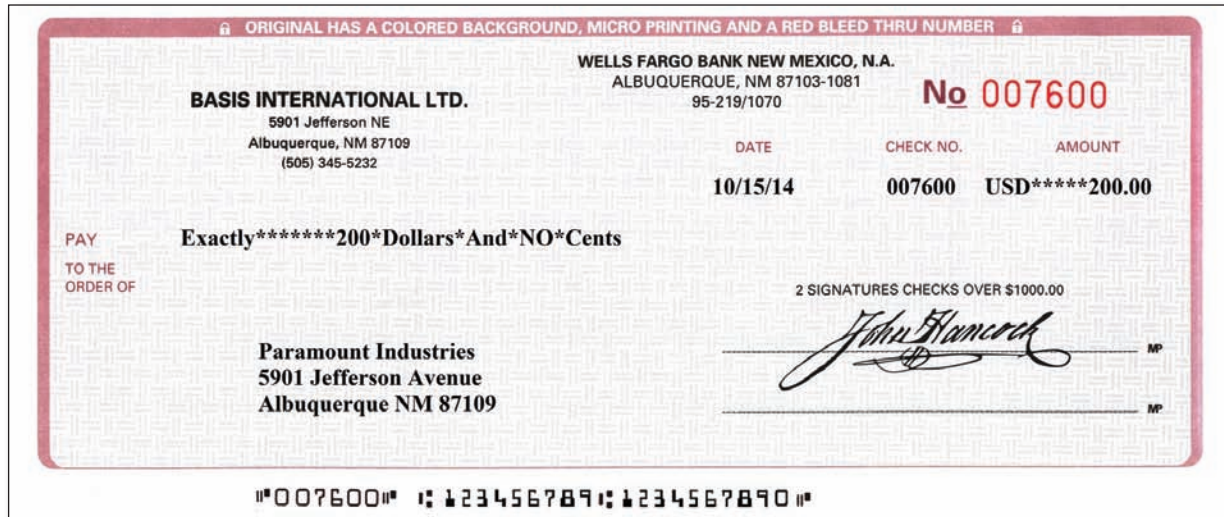


Figure 5. Check with digital signature applied

Summary

The AddonSoftware Accounts Payable Payment Authorization feature delivers several important benefits. It improves record keeping by storing images of invoices online attached to the invoice record, facilitating instant invoice retrieval. Paper invoices can be shredded and discarded saving file space and paper handling. The database stores invoice approvals, preserving the approval and check signing history of all invoices. Workflow is streamlined as each responsible person in the process receives notification via email when their attention is required and all others involved in the process are informed of the status. Most importantly, executives can perform the approval process from anywhere they may be, whether that is in the office, at home, or on the road. Who wouldn't want to save time by limiting the movement of paper, eliminating the filing of paper invoices, and eliminating manual check signing? It is now as painless as payments can be. ■



Don't Put All of Your Jetty Eggs in One Context

For those of you who like to look for the changes that BASIS makes in each BBJ® upgrade, you may have noticed in version 14.12 a new file called `jetty.xml`. This addition wasn't, as some may think, a stealthy way for us to migrate our configurations into xml but rather a way to provide much greater flexibility in how to deploy your applications and pave the way for things to come. With this change, the benefits reach a wider audience. This article takes a closer look at the benefits, and will help you understand the added power we're providing and help you modify your deployments to take advantage of this power.

Configure Jetty the New Way

So why did BASIS decide to confuse users with yet another configuration file?

You may not have even thought about what is going on behind the scenes – you write your application and you configure it in Enterprise Manager (EM) to be accessible via a web browser. Perhaps you'll run it as a BUI application, or use Java Web Start/JNLP to launch it on your desktop via the Web, but either way, you will deploy it within the internal Jetty web server. However, there is only one server that is making all of your applications available and that means all of your applications are visible to all of your users. We decided that it was time to enhance that.

Let's start by looking at the new `jetty.xml` file that is behind the configuration. **Figure 1** shows the default file that BBJServices created automatically for you.



Richard Stollar
Software Developer

```
<?xml version="1.0" encoding="UTF-8"?>
<jetty-config>
  <contexts>
    <bbj docbase="$basis_home/htdocs" path="/" >
      <welcome-file>index.html</welcome-file>
    </bbj>
    <bbjem path="/bbjem"/>
    <resource name="htdocs" path="/files"
      resource="$basis_home/htdocs" />
    <resource name="documentation" path="/documentation"
      resource="$basis_home/documentation" />
    <resource name="baristahelp" path="/BaristaHelp/WebHelp"
      resource="$basis_home/barista/sys/help/WebHelp" />
    <resource name="addonhelp" path="/AddonHelp/WebHelp"
      resource="$basis_home/apps/aon/help/WebHelp" />
  </contexts>
  <admin user="admin" password="B1NhCfk1XmL0/u1WA8aoKQ=="
    encrypted="true"/>
</jetty-config>
```

Figure 1. The beginning of the `myServlet` class

Context is the key word here. A context is a place where an application exists. Remember that these applications are BUI, Web Start, web services, or web servlets, but currently there is only one context at `http://{server}:8888` or `http://127.0.0.1:8888`. All of your applications are in that context, but they don't have to be. BASIS provides a way for you to create new contexts and decide which applications are available in which context.

The Root BBj Context

The standard `jetty.xml` defines several contexts, the most significant of which is the BBj or root context that by default contains all of the applications. When you deploy an application – BUI, JNLP, or servlet – you can specify a custom name for the context; if you don't specify a custom context, then Jetty will assume the root context. Keep this root context in mind as you read through this article as we'll mention it from time to time.

Hostnames and IP Addresses

So let's step back a little bit and think about the hostname and what it means. When you open a page in your browser, you likely use a nice friendly name like `google.com` and the browser looks up `google.com` to find its IP address. Read on for the details of how that works or if you are already familiar with IP addresses, skip to the next section in this article 'Hostnames and how BBj Services Handles Them'.

The browser is the client and the server is, well, it's the server. All clients need to obtain an IP address for the server and this is done using hostname resolution. If the client doesn't already know the IP address for the server (it can be listed in a special hosts file), then the client uses a domain name server (DNS) to look up the IP address of the server.

The browser communicates with that address using the HTTP protocol and all requests have a header that contains the name of the server you want to talk to, `google.com` in our case. When the server receives the request, it looks at the header to decide if and how it will deal with that request. For example, `www.google.com` will go to the search engine we're all familiar with, whereas `translate.google.com` will go to the Google translator application. It could be that both of these friendly names resolve to the same IP address or it could be that they don't, but what's important is that it is the server's job to decide if and indeed how it will respond.

Hostnames and how BBjServices Handles Them

Your machine running BBjServices exists on a particular IP address, `10.0.0.10` or `192.168.0.10`, for example. It will most likely have at least one hostname that should be known by the clients, `jupiter` for example, but you can use almost any hostname.

In addition to a specific hostname or IP address, your computer has what we call a *loopback device*. The loopback device is most commonly referred to as `localhost` or by the IP address `127.0.0.1`.

Therefore, by default any hostname that resolves to the IP address of your server gains access to the root context we were talking about earlier. The URL `http://jupiter:8888/` will display the BBjServices welcome page. All of your Web Start applications will be in `/jnlp/*`, your web services in `/webservice/*`, your servlets in `/servlet/*` and your BUI applications in `/apps/*`.

Creating Custom Contexts

Keeping all of your eggs in the same basket, or in the same *context*, is easier to administer but does not suit everyone. Suppose you have some applications that your customers access and another set of applications that your internal staff use. You may need to create a level of separation between them so that the external users cannot access the internal applications even if they can correctly guess their names. What you need to do is tell Jetty what hostnames it should accept and which of your precious applications live there. Here's how we go about it.

Consider that you are developing a servlet-based application that requires the deployment of multiple servlets, but at the same time, you want to restrict this application to a specific host or sub-domain. To achieve this goal, create a new context in `jetty.xml` with the basic XML content as shown in **Figure 2**.

```
<context name="myapp" path="/" docbase="\var\www\myapp" >
  <host>myapp.juniper</host>
  <welcome-file>index.html</welcome-file>
</context>
```

Figure 2. A custom context

Be sure to insert this context XML element immediately before the `</contexts>` marker.

Now, when you restart BBjServices, you will have a new context available called `myapp` into which you can deploy your servlets. We assigned the new context a `<host>` element that tells Jetty the hostname for the context. Your servlets, for example, will be available through `http://myapp.juniper:8888/servlets/*`. The `\var\www\myapp` folder specified in `docbase` is where static content such as images or html pages should be saved.

Deploying Your Servlets

If you've created a servlet-based application, you'll know that keeping the servlet deployed is extremely important. You will most likely dedicate a BBJ program to deploying the servlets and add this servlet deployment program to the autorun list.

You will have to make a small change to your servlet deployment program in order to deploy your servlets to a specific context. Currently, your servlet deployer will contain a line of code similar to this:

```
registry!.publish("some_path", myServlet!)
```

If you leave this code alone, then Jetty will deploy your servlets in the root context as before. However, to specify a context, add its name as the first argument like this:

```
registry!.publish("myapp", "some_path", myServlet!)
```

Once deployed, users will only be able to access your servlet through that context. Creating your own custom contexts also ensures that sessions are unique and that session data for servlets in one context remains isolated from servlets in another context. Even if you deploy the same servlet to two different contexts, they will not share session data. The same client can access both servlets without corrupting data.

To simplify deployment, you can specify the servlets to register within a context when Jetty starts. To achieve this, add a servlet entry to your custom context. **Figure 3** shows a servlet entry that will make your servlet available on `http://myapp.juniper:8888/servlets/myservlet`.

```
<context name="myapp" path="/" docbase="\var\www\myapp">
  <host>myapp.juniper</host>
  <welcome-file>index.html</welcome-file>
  <servlet name="myservlet"
    program="MyServlet.bbj"
    config="/usr/bbj/cfg/config.bbx" />
</context>
```

Figure 3. Adding a servlet to the custom context

You can add as many servlet entries as required to the context and they will deploy automatically each time your BBJServices restart. Each servlet entry requires three pieces of information –

1. The **name** of the servlet as it will be used in the URL (in this case, `myservlet`).
2. The **program** file name that should be located in some path in your `config.bbx` file's prefix (in this case, `MyServlet.bbj`).
3. The **config** file used when executing the servlet (in this case, `C:\BASIS\cfg\config.bbx`); if you do not specify a config file, `myservlet` will use the default `config.bbx`.

In order for the automatic deployer to function, you need to set the admin entry in `jetty.xml` by specifying your admin user's username and password.

Specifying the Admin User's Username and Password

If you change the username or password that you use for your admin user, then you will need to update the admin entry in `jetty.xml`. To set the user credentials, edit `jetty.xml`, enter the correct user and password in the admin entry, set the *encrypted* flag to false and save the file. It would need to look something like this:

```
<admin user="admin" password="admin123" encrypted="false" />
```

Don't worry about entering the password in clear text because BBJServices will encrypt it automatically when they start and after restarting BBJServices, the entry will look more like this:

```
<admin user="admin" password="B1NhCfk1XmL0/u1WA8aoKQ==" encrypted="true" />
```

Deploying Applications to a Custom Context

Another element that you may wish to control is the deployment of your applications. Up until now, your applications have all lived in the same place, but in line with the other changes we've put in place we decided to give you greater control and flexibility of that too. You can specify a custom context for all your applications, that's BUI and Web Start/JNLP applications as well as web services. In much the same way described above for servlets, you can control which applications Jetty places in which context or you can again leave the deployment to the root context. The EM allows you to select a context from the list of available contexts and restrict access to a specified hostname within that context.

If editing configuration files is your thing then you'll know that all BUI and Web Start applications are configured through `bui.ini` and `jnlp.ini`, which are both in the `basis_home\cfg` folder and these files list the applications deployed. Inside each applications' configuration you can add a `CONTEXT={context_name}` to control which context the application will be deployed to. For example, `CONTEXT=test` will deploy the application to the test context. Each web service also has its own configuration file, which

can be found in the `basis_home\cfg\webservices` folder and again you can specify the context by adding `context={context_name}` to the file.

For those of you that don't want to mess with the configuration files directly, then the EM is the place to go. For each of the different application types you can configure the context where the application will go.

Whether you configured contexts with the EM or you edited the configuration files manually, the result is the same; you've placed your application within that custom context. Remember to define the custom context before assigning an application to it, or the application will be invisible and you really don't want that!

Figure 4 shows setting the context in action for web services, but the same principle applies on the BUI and JNLP configuration pages. When contexts have been defined in `jetty.xml` you can select which context the application will be deployed to from the dropdown list.

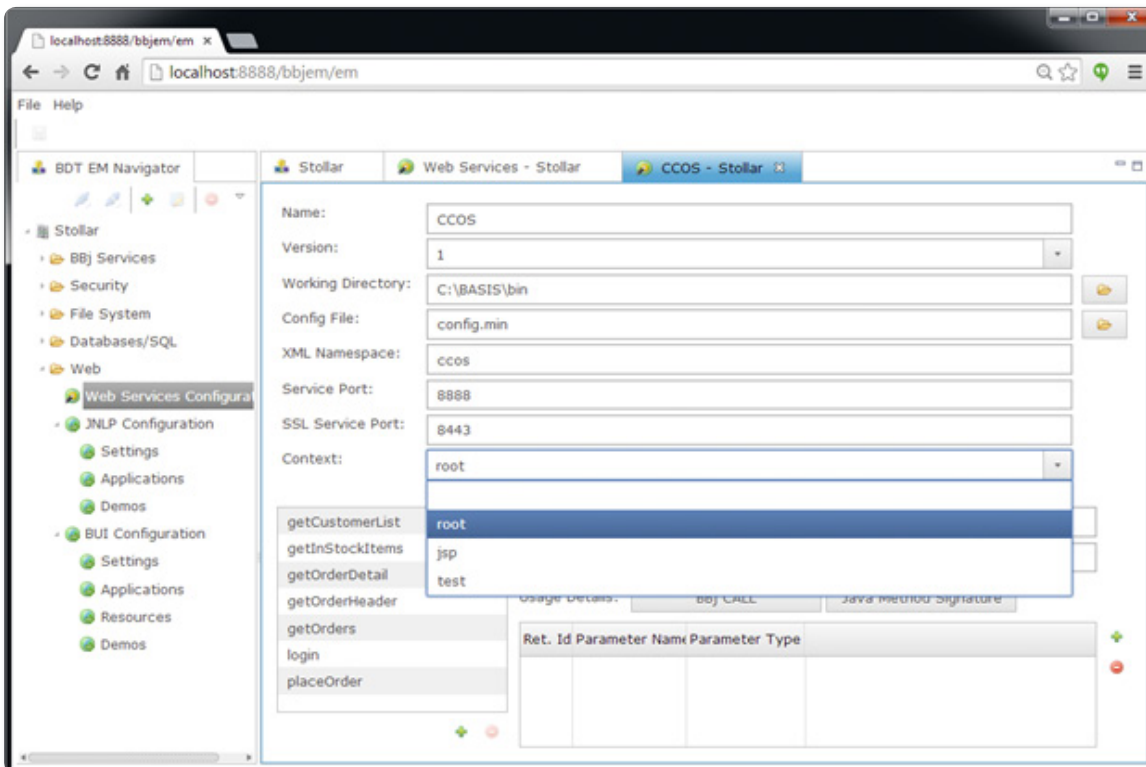


Figure 4. Set the context for your applications

Why did we go to all That Trouble?

We listen to the newsgroups and recently received the question, *"Is there any way to turn off the browser EM access without disabling our JNLP access?"* The problem was that everything was accessible on the server and you couldn't do much about it. Unless you took special action by installing another web server like an Apache server to filter requests, all your applications as well as EM were accessible.

Besides allowing you to restrict access to your applications, another significant benefit relates to BBJ Servlets. Servlets are by design stateless but you almost certainly need to maintain some kind of user state that you would achieve through the underlying session. The session ID cookie identifies the client's session with each request but with all your servlets on the same URL (`http://jupiter:8888/servlet/*`), then there is no real way to have some servlets use a different session from other servlets. Using contexts allows each session to maintain a unique session ID and thus unique session data.

Keep Enterprise Manager Safe

One of the applications that you may want to take control over is EM. You don't want someone trying to hack into your EM that resides on the same server as your public application, do you? Keep that employee you had to let go last week from acting out his need to mess with your server or delete your database.

Ask yourself two simple questions, *"Is my Enterprise Manager accessible over the Internet?"* and *"Do I still have **admin123** as my admin password?"* For many of you, the answer to both of these questions will be, "Yes!"



You can take control over the EM context and assign it to a specific hostname, **localhost** for example, and in doing so, ensure that the EM is only accessible on the physical machine or to a user logged in with remote desktop or a VNC connection. **Figure 5** shows a modified entry in **jetty.xml** that limits EM access to localhost.

```
<bbjem path="/bbjem" >
  <host>localhost</host>
</bbjem>
```

Figure 5. Restrict Enterprise Manager access

The example in **Figure 5** tells Jetty to allow access to the EM only through the specified hostname, **localhost**. You cannot even access it using the IP address unless you add another host entry such as `<host>192.168.0.99</host>`. How cool is that?! In addition, the hostname can be a subdomain like **bbjem.juniper.com**.

OK, so it's clear that you can set a hostname for your application contexts but...

What is the Path all About?

In addition to the hostname, a context has a path that really means the top of the context. We can create several contexts that are all accessible through the same hostname but have different path entries. Look at **Figure 6** and examine this more closely.

```
<context name="myapp" path="/" docbase="\var\www\myapp">
  <host>myapp.juniper</host>
  <welcome-file>index.html</welcome-file>
  <servlet name="myservlet"
    program="MyServlet.bbj"
    config="/usr/bbj/cfg/config.bbx" />
</context>
```

Figure 6. Contexts with a path

These context entries are the standard entries created automatically for you. Because none of them have a specific `<host>` entry, they are available to all hosts but the **path** is different providing **http://juniper:8888/files**, **http://juniper:8888/documentation**, and so on.

Earlier, you saw how **Figure 5** showed adding a hostname to the EM configuration to limit access to a specific hostname but the EM also has a path that is **/bbjem**, but again it doesn't have to be and it is easy to place EM on **http://em.juniper:8888/em** by using **path="/"** instead of **path="/bbjem"**.

Using Context Parameters and Attributes

When you start using contexts to control your applications, you may find the need to store context specific data. There are two types here, *parameters* and *attributes*. Parameters are defined in **jetty.xml** and are read-only to your servlets Whereas attributes can be created, edited, and destroyed on the fly within your servlet code. **Figure 7** shows how to define parameters in **jetty.xml**.

Your servlets have access to these parameters and attributes through the new **BBjJettyContext** object. The use cases are beyond the scope of this article but the examples in **Figure 7** are fairly self-explanatory.

```
<context name="myapp" path="/">
  <param name="database" value="myDatabase" />
  <param name="currency_code" value="USD" />
  <param name="currency_symbol" value="$" />
</context>
```

Figure 7. Context parameters in jetty.xml

The **BBjJettyContext** class provides access to the parameters and attributes within the context. You can obtain a **BBjJettyContext** class from the **HttpSession**, and the code sample in **Figure 8** demonstrates how to read a parameter in your **BBj Servlet**.

Remember that parameters can only be read by your servlets and if you need to store information in the context, then you should use attributes. Attributes stored in the **BBjJettyContext** are available to all servlets in your application, and are shared between requests and sessions. That means that the attributes are globally available to all visitors of the web application whereas session attributes are just available to a single user.

```
request! = p_event!.getBBjHttpRequest()
session! = request!.getSession()
context! = session!.getContext()
currencySymbol$ = context!.getInitParameter("currency_symbol")
currencyCode$ = context!.getInitParameter("currency_code")
```

Figure 8. Reading application context parameters

Using Java Elements

We have opened up the Java world too as Java provides some features that you may find useful. To incorporate Java elements into your application context, create a session classpath through the EM in the usual way and specify the classpath in the context by adding a classpath element as shown in **Figure 9**.

```
<context name="myapp" path="/" classpath="myappClasspath" >
  ... other elements ...
</context>
```

Figure 9. Adding a classpath to the Context

You can add third party Java servlets, context listeners, and request filters directly into your Jetty Server without having to use another application server. **Figure 10** shows an example of how to setup Java classes.

```
<context name="myapp" path="/" classpath="myappClasspath" >
  <j-listener class="com.acme.SystemInitializationListener" />
  <j-filter class="com.acme.Filter" filter="/*" />
  <j-servlet class="com.acme.BusinessProcess" mapping="*.bp" />
</context>
```

Figure 10. Adding Java elements to the Context

These are common elements of a Java web application. The specifics about how to write them and what you would use them for are well beyond the scope of this article. But when the need arises, they can be incorporated seamlessly into a custom context. Add a `<j-servlet>`, `<j-filter>`, or `<j-listener>` entry to define the components.

Sometimes a Java servlet needs initialization parameters and these can easily be added to the `<j-servlet>` tag as shown in **Figure 11**.

```
<j-servlet class="com.acme.BusinessProcess" mapping="*.bp">
  <param name="com.acme.dateFormat" value="yyyy MMM dd" />
</j-servlet>
```

Figure 11. Passing initialization parameters to a Java servlet

Summary

By introducing contexts, BASIS now allows you to take better control of how you deploy your Jetty Web Server applications by supporting multiple hosts and providing ways to restrict access to applications. Nevertheless, the good news is that you don't have to do anything unless you feel the need. Leave `jetty.xml` alone and everything will continue working the same as it has always done up until now. However, even if all you do after reading this article is restrict access to your EM, then I will consider this article to have been a great success. ■



Download and run the [code samples](#)

BARISTA®

Gets you (re)productive in no time!



RABBIT APPLICATION DEVELOPMENT





Wash up With SOAP Web Services

Now with rich data structures and authentication

B BBJ® web services received a facelift, which allows you to build web services that handle more complex data structures as well as collections of records. This article presents a brief tutorial on how to create a SOAP web service that uses custom Java types in the parameter set to extend your web service offerings. We have also added Basic authentication access and will show you how to use it. At the end of this article you will find an URL where you can download the examples.

An Overview on Enhanced Data

You can design a typical BBJ web service to have parameters of the standard BBJ data types BBJString, BBJNumber, and BBJInt, but you may require a greater variety of types. Enhancements to BBJ 14.11 now make it possible to create a web service that uses Java types on the parameter list, providing a richer web service capability.

Let's suppose that we want to publish a web service that deals with customers, but the customer record is rather complex, and the main customer record has sub records for the address and the contacts. **Figure 1** shows how this might look.

In our structure, *Address* and *Contact* are complex subtypes, and there could be many contacts records per customer. Okay, so maybe it's not that complex but it does serve to illustrate the point.

Customer		Address		Contact	
BBJInt	ID	BBJString	Street	BBJString	First Name
BBJString	Name	BBJString	City	BBJString	Last Name
Address	Address	BBJString	Zip Code	BBJString	Job Title
BBJVector	Contacts	BBJString	State	BBJString	Phone
				BBJString	Email

Figure 1. Record structure for Customers

To achieve our goals of having a web service that can deal with this data structure, we're going to have to complete several steps.

1. Create the main web service BBJ application.
2. Configure the web service in Enterprise Manager.
3. Create a set of JavaBeans that represent the data structure.
4. Configure a classpath for the Java classes.

Additionally, we're probably going to need to do the following:

5. Create utility methods for filling the Java data structures.



Richard Stollar
Software Developer

Before we get into all of that, let's look at how the web services work to understand why we need to do this.

BBj Web Service Classes and Deployment

BBj web services rely on *wsgen*, which comes with Java. The *wsgen* tool parses the web service implementation class and generates the required files for web service deployment.

For *wsgen* to function, it needs to have access to the required Java classes through a classpath that's used during execution.

You have two real options for how to provide *wsgen* with the classes needed.

Option 1: Create a .jar file that contains the classes.

Option 2: Add the folder where the classes were generated in the classpath.

For many reasons, including the fact that a .jar file is far more portable than a whole bunch of class files, Option 1 is the preferred method when dealing with deployment. However, during the development phase, Option 2 is easier as you don't need to recreate the .jar file whenever the code changes.

So, you'll have a project in Eclipse that holds the Java source files and the corresponding class files will be generated into a bin folder, and this is what we're interested in. For example, `C:\Work\Examples\JavaBeans\bin\`, but it may vary for you based on the location of your project's bin folder.

Crossing the Bridge from BBj to Java Web Service

When you develop your web service in BBj, there's a certain amount of magic that goes on behind the scenes. Your BBj program, however simple or complex, needs to be wrapped up with a Java implementation. Through Enterprise Manager, you specify the prototypes for your web service's methods. These prototypes are used to generate a Java web service for your BBj code. That front-end web service is sent to *wsgen*; as mentioned earlier in this article, to generate all the classes. Finally, the web service is deployable.

Creating the Main Web Service

Writing the web service's code is beyond the scope of this article and much depends on your requirements. In general terms, it is a BBj program with one or more entry points that each perform some part of the web service's functionality.

Figure 2 shows a small piece of BBj code for a `getCustomer()` method in a web service.

```
getCustomer:
  enter customerId%, customer!
  customer! = new Customer()
  customer!.setCustomerId(1)
  customer!.setName("ACME")
  address! = new Address("1 The Street", "The City", "NJ", "52444")
  customer!.setAddress( address! )
  vect! = new BBjVector()
  barny! = new Contact("Barny", "Rubble", "Rock Breaker", null(), "barny@bedrock.com")
  fred! = new Contact("Fred", "Flintstone", "Site Foreman", null(), "fred@bedrock.com")
  vect!.add ( barny! )
  vect!.add ( fred! )
  customer!.setContacts( Util.makeContactArray(vect!.toArray()) )
  exit
```

Figure 2. Sample web service code

This sample creates a customer record with static data, adds the address, and creates a vector containing two contact records which it then converts to an array using a utility class and adds to the customer record. It completes a record with fixed data whereas your web service will most likely be database driven.

Configure the Web Service

Configuring the web service is done through Enterprise Manager as normal, at least up until the point where you need to specify the parameters for your service methods. Rather than selecting the type from the dropdown list you can enter the fully qualified name



of your class as the parameter type. In our example the fully qualified class name is `com.acme.beans.Customer`. **Figure 3** shows this in action.

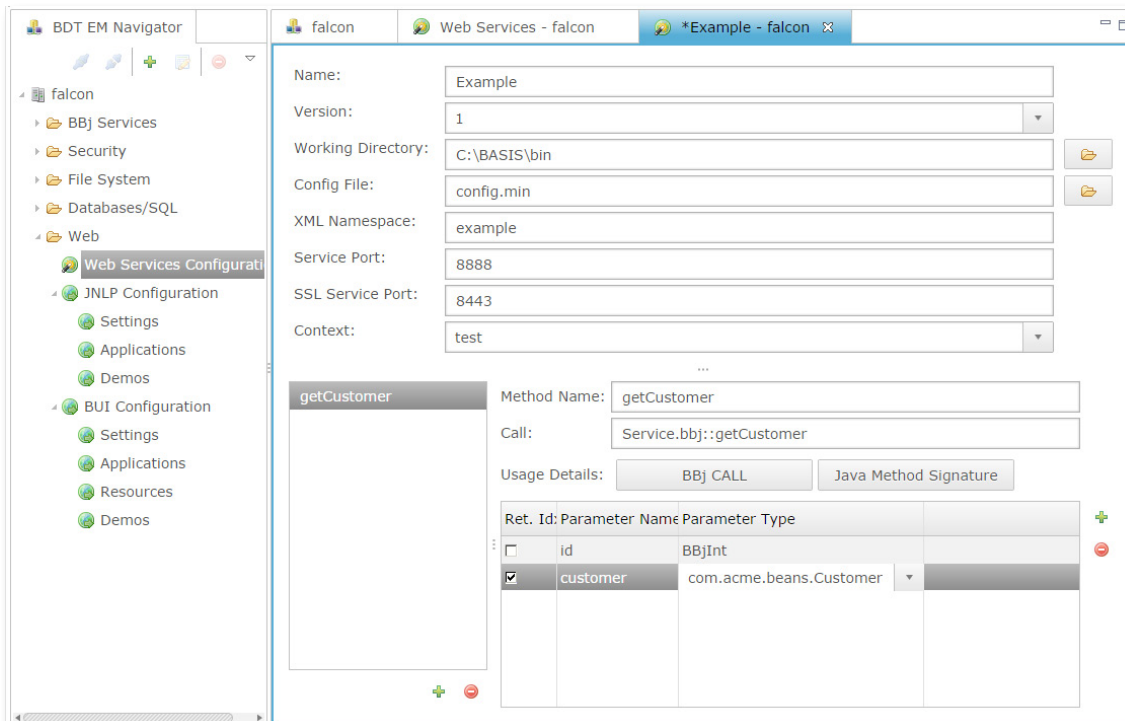


Figure 3. Specifying a custom class

Creating the Java Classes

We need to create a set of JavaBeans that represent our data structure. A JavaBean is a special type of class that encapsulates many objects into a single object (the bean) and, unless your classes conform to the JavaBeans specification, you won't be able to use them in your web service.

Again, I don't want to go too deep into explaining how you should go about creating JavaBeans as the downloadable example should give you all you need. It's beyond the scope of this article to cover how you should write the JavaBeans for the data, just remember that the classes must be serializable, have a zero-argument constructor, and allow access to properties using accessor (getter and setter) methods; this makes them a bean. **Figure 4** shows sample BBj code for the Customer record.

```
package com.acme.beans;

import java.io.Serializable;

public class Customer implements Serializable {
    private Integer m_customerID;
    private String m_name;
    private Address m_address;
    private Contact[] m_contacts;

    public Customer() {
        // Mandatory zero-parameter constructor.
    }

    public Customer(Integer p_customerID, String p_name,
        Address p_address, Contact[] p_contacts) {
        m_customerID = p_customerID;
        m_name = p_name;
        m_address = p_address;
        m_contacts = p_contacts;
    }

    public Integer getCustomerID() { return m_customerID; }
    public void setCustomerID(Integer p_customerID) { m_customerID = p_customerID; }
    public String getName() { return m_name; }
    public void setName(String p_name) { m_name = p_name; }
    public Address getAddress() { return m_address; }
    public void setAddress(Address p_address) { m_address = p_address; }
    public Contact[] getContacts() { return m_contacts; }
    public void setContacts(Contact[] p_contacts) { m_contacts = p_contacts; }
}
```


Setting Up the Classpath

You can choose to add a .jar file to your classpath through Enterprise Manager. To do this:

1. Pick BBj Services > Java Settings from the left menu.
2. Select the 'Classpath' tab.
3. Click the [+] icon above the 'Classpath Names' list.
4. Enter a meaningful name for the classpath entry; I chose **service**.
5. Click [OK] to save it.
6. Ensure that your newly created classpath entry is selected and then click the [Add a Jar] icon above the 'Classpath Entries' list (to the right).
7. Locate the jar file and select it with a double-click or by selecting it and clicking [Open].

Finally you need to add the same entries to the <default> bbj classpath entry and here's how:

8. Select the <default> classpath entry in the list of classpath names.
9. Click the [Add a Jar] icon above the 'Classpath Entries' list, locate the jar file and select it as before.

Alternatively, you can manually add the classes folder to your classpath in your **bbj.properties** file. It's important to remember that all classpath entries must begin with **basis.classpath**. Here is an example of what it should look like:

```
basis.classpath.service=C:\\Work\\Examples\\JavaBeans\\bin
```

Adding the Classpath to the Context

BBj 14.11 introduced contexts for the deployment of your applications, which is outside of the scope of this article but there is more information in this issue's *Don't Put All of Your Jetty Eggs in One Context*.

You'll need to decide the context to which we're deploying our web service and add the appropriate classpath element to its configuration in **jetty.xml**. For now, let's deploy our web service to the main bbj root context and add the classpath element to the context entry shown in **Figure 5**.

```
<bbj docbase="$basis_home/htdocs" path="/">
  <classpath>service</classpath>
  <welcome-file>index.html</welcome-file>
</bbj>
```

Figure 5. Adding the classpath to the context

Adding the classpath to the context is important as the web service generation will fail without it because this classpath entry is passed to **wsgen**.

Dealing With Collections

Our **Customer** record has many **Contact** records. From BBj's perspective, these contacts are held in a **BBjVector** but when you want to fill the Java object with the contents of the **BBjVector**, you'll need to transform the data into an array of objects of the specific class. Enter the need for a utility class I mentioned earlier that will handle this.

Using a utility class on the Java side can be a valuable tool as you can create all sorts of standard methods for converting data. When you have a collection of objects you'll probably need to have a suitable converter similar to the one shown in **Figure 6** which transforms an array of objects into an array of **Contact** objects.

```
public class Util
{
    public static Contact[] makeContactArray(Object[] objects)
    {
        return Arrays.copyOf(objects, objects.length, Contact[].class);
    }
}
```

Figure 6. Converting an array of objects into an array of **Contact** objects

Your BBj program will use this method to convert the **BBjVector** into an array of **Contact** objects as follows:

```
customer!.setContacts(Util.makeContactArray(vect!.toArray()))
```

Add Basic Authentication

Authenticating clients to a web service can be a rather involved process but thankfully doesn't have to be. If you use the HTTPS protocol for deploying your web services then Basic authentication (introduced for preview in 14.12) should be sufficient and it is widely used. The username and password are encoded in the HTTP request header and then validated server-side.

Applying validation to your web service requires you to write a custom routine in your web service implementation which has three parameters. The input parameters are username, password and a response to indicate if validation was successful or not. **Figure 7** shows a simple implementation, but in a real world you might be validating the user through a database or some other source.

```
myAuth:
  enter user$, pass$, auth%
  auth%=0
  if user$="admin" AND pass$="admin123"
    auth%=1
  endif
  exit
```

Figure 7. Example authentication routine

The final step in this process is to configure the authentication routine through Enterprise Manager. Simply enter the name of your authentication routine in your web service's 'Authentication' field as shown in **Figure 8**. In this example, that would be **myAuth**. Finally, deploy your web service and away you go!

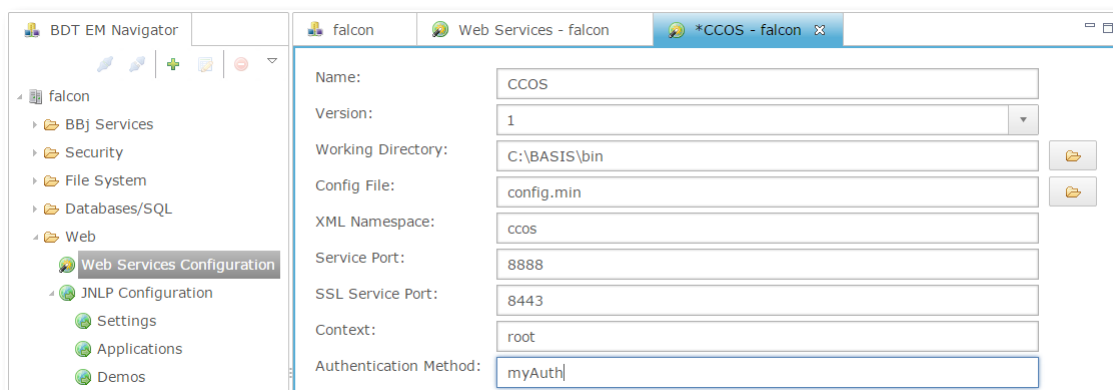


Figure 8. Setting the authentication in Enterprise Manager

Configuring the client application is beyond the scope of this article, but most SOAP test tools provide ways to supply credentials for Basic authentication.

Remember that Basic authentication sends the username and password in a non encrypted form. It is by itself very insecure, but becomes secure when used in conjunction with the HTTPS protocol.

Summary

As you have read, creating more complex web services is much easier as you can now take advantage of rich SOAP web services to send and receive complex data structures as well as collections of records. Your web services will be easily accessible to a range of client technologies and conform to well-known standards. ■



- Check out the example available for download at www.basis.com/14code
- Read *Don't Put All of Your Jetty Eggs in One Context*
- Download and run the [code samples](#)





By Amer Child
Digital Communications/
Web Developer

On-Demand Enlightenment at the BASIS E-Learning Center

Business BASIC is only as effective as the developer using it, hence our continued commitment to strengthen developers' skills. Through the years, we have provided a variety of opportunities to enlighten and equip developers to use our tools so they can be successful. Keeping pace with ever-changing technology, BASIS has hosted a myriad of these learning opportunities for developers skill growth:

- **Reference materials** - [online Help](#), [Knowledge Base articles](#), and [The BASIS International Advantage](#) publication
- **Live** - classroom training, TechCon conferences, and regional TechView seminars
- **Virtual** - [Java Breaks](#) and hosted trainings

BASIS Presents a New Opportunity!

The **BASIS E-Learning Center**, a new educational opportunity is now available dynamically across all time zones. This self-paced, *on-demand* training solution meets your educational needs wherever you are, which is an ideal way for you to achieve the necessary training to become most effective in your trade. After all, who wouldn't want to be successful providing well-tuned, user-friendly, robust solutions that meet your customers' precise needs and generate a sustainable revenue stream?

Go to elearning.basis.com and take a test drive with the Training Prep course to see how this portal works. Look at any of the 15 courses and their descriptions that cover each building block – Language, Development Tools, Database Management, System Administration, or all the bundles in our ERP building block solution, AddonSoftware®. The most popular courses thus far have been the AddonSoftware training courses; the other very well-attended courses are *Report Writing With Jaspersoft Studio* and *BBJasper and Barista® Application Framework*.

Come and go as you like with the time you have available in your busy schedule, using the computer and Internet connection of your choice ... from home or office, on the road, or at your local coffee shop.

While E-Learning courses are priced the same as our online hosted courses, with free attendance to Preferred Partners, E-Learning offers an additional advantage by giving you direct control of your experience. You can start the course when you wish, according to your schedule and at your own pace, pausing and playing back at will for a truly personal experience. You can also post any questions that arise during your training to a monitored online discussion forum.

The Buzz

Feedback on the E-Learning experience has been superb. Marco Poblete, President of Poblete Consulting Services LLC, whose company had an early test run with the portal and shares their very positive experience. *"BASIS' new E-learning portal is a great resource to our company as we evolve our line of services and products. The portal allows us to learn how to improve the look-and-feel and efficiency of legacy BBx applications."* Poblete continues, *"Having the opportunity to schedule our training time at our convenience enables us to continue working with our customers while learning the new features of the BASIS Product Suite. The greatest advantage is that we can review the session whenever we want at a time and pace that is most convenient to us!"*

After you have tried one of our on-demand E-Learning courses, please share your feedback with us.

Looking Ahead

As BASIS continues our commitment to training, we will add more online courses as well as schedule hosted courses for those who still prefer real-time interaction with the trainers. Whether you are new to the BASIS community or just want some strengthening and enlightenment, visit the E-Learning Center to find the course that is right for you. Whichever is your preference, come-and-go or hosted online training, be on the lookout for new offerings to help you become a more efficient and stronger developer.

Summary

If taking time out of your busy schedule to sharpen your skills has been difficult in the past, we have removed that hurdle. E-Learning meets you where you are and eliminates the scheduling conflicts and time allotments that may have been obstacles in the past. There's always value in knowledge, so start strengthening your skills today with E-Learning! ■



Check out our training offerings

- [E-Learning Center](#)

- [BASIS Training](#)

Visit our online reference material and other resources

- [Online Help](#)

- [Knowledge base articles](#)

- [The BASIS International Advantage](#)

- [Java Breaks archives](#)

Try it Out

1. From elearning.basis.com, click 'Training Prep'.

The screenshot shows the BASIS Events Calendar and Course categories page. The calendar displays events for the month of October, with a specific event '10am Java Br...' on October 14th. The Course categories list includes Training Prep (1), ERP Building Blocks, AddonSoftware, Development Tools, Barista Application Framework (1), Jaspersoft Studio (1), Language (1), Database Management (1), and System Administration (3). A red arrow points to 'Training Prep (1)' in the list.

2. Click 'Enroll Me', 'Log in as a guest'.

The screenshot shows the BASIS Training Prep - Enroll Me page. The page displays the 'Training Prep - Enroll Me' button, which is highlighted with a red arrow. Below this button is a 'Log in' form with fields for 'Username' and 'Password', and a 'Log in as a guest' button. The 'Log in as a guest' button is also highlighted with a red arrow.

3. Follow the instructions listed for the course!

Training Prep

Complete the steps below to prepare your system for training.

To guide you through these steps, view the 15-minute video by clicking the [Training Prep Video] button below.

1. Ensure you have a JDK installed at the correct version your BBj supports (Java version 1.7 u51 or higher).
2. Download and install the latest release from links.basis.com/getproduct; choose installation option 5 "BBj + Barista + BASIS IDE + AddonSoftware + Demos".
3. During installation, select the 30-day demo license and change the user count to 5.
4. We deliver our Integrated Development Environment (IDE) as a plug-in to Eclipse, the most widely used Java IDE. In preparation for using this tool, including our system administration tool the BBj Enterprise Manager (EM). From links.basis.com/eclipse, select the Eclipse URL at the top of the page. From the resulting eclipse.org web page, select your operating system link and select a mirror link to download and install Eclipse.
5. From links.basis.com/eclipse, install the Business BASIC Development Tools and the BBj Enterprise Manager.
6. If you are attending a hosted course over the Internet via Adobe Connect, test your system before the live prep session at links.basis.com/connect-test.
7. For AddonSoftware courses (only) -- Watch the 10-minute "Quick Copy" video (link below) to step through the process of setting up your AddonSoftware environment.
8. If you have any issues email "training@basis.com"

Training Prep Video

Quick Copy Video

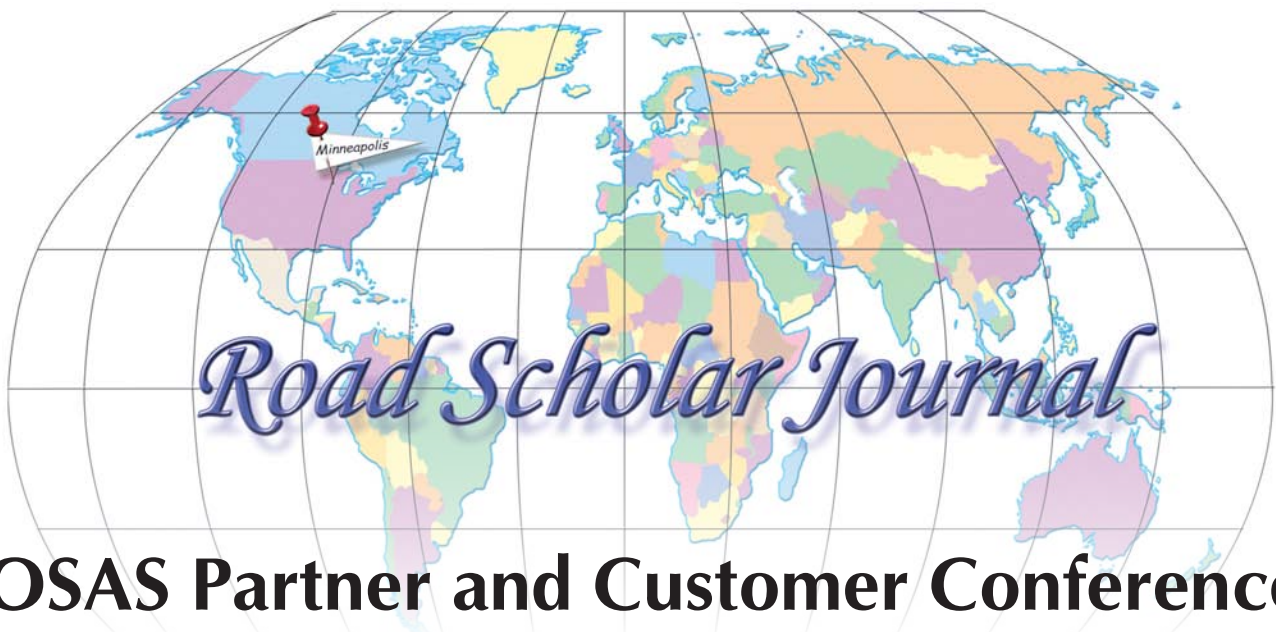
Before you can enroll in the next class, create an account as prompted. Once you have an account, simply log in to enroll in future courses.

? Is this your first time here?

Hi! For full access to courses you'll need to take a minute to create a new account for yourself on this web site. Each of the individual courses may also have a one-time "enrolment key", which you won't need until later. Here are the steps:

1. Fill out the [New Account](#) form with your details.
2. An email will be immediately sent to your email address.
3. Read your email, and click on the web link it contains.
4. Your account will be confirmed and you will be logged in.
5. Now, select the course you want to participate in.
6. If you are prompted for an "enrolment key" - use the one that your teacher has given you. This will "enrol" you in the course.
7. You can now access the full course. From now on you will only need to enter your personal username and password (in the form on this page) to log in and access any course you have enrolled in.

Create new account



OSAS Partner and Customer Conference

August 11-14, 2014 marked the return of the Open Systems, Inc. Partner and Customer conferences. After many years in Las Vegas, OSI decided to bring the conference back to their home in beautiful Minneapolis, MN, at a new Radisson property adjoining the Mall of America. With over 520 stores, an indoor theme park, Legoland, comedy club, and many other attractions, the mall provided a great venue for after conference hours entertainment and additional options for entertaining conference attendees. Per Wikipedia, the mall receives more than 40 million visitors annually, the most of any mall in the world. It was impressive!

Dr. Michael Bertini (Owner/CEO) opened the conference with a message on embracing change. Dr. Bertini's message states, *"Both competition and technology are pushing users to re-think their business processes. The need to mobilize and collaborate beyond the back office is creating a significant shift within our client base. Not only are they embracing change, but they are pushing us to provide innovative products and services. This clearly challenges us to become much more agile to effectively deliver products and services that will meet our clients' ever-changing objectives."*



BASIS understands the ever-changing technology world, and the need to meet these challenges every step of the way. Our perseverance is constantly providing our developers with more deployment options, tools, utilities, and wizards. These continual additions are proof of our commitment to the community to meet the developers' and users' ever evolving needs.

Our sessions featured the BASIS Dashboard Utility that BBJ 14.0 includes at no additional cost. Showing the developers how to easily implement the Dashboard in any version of OSAS or embed a dashboard widget into most any application generated a lot of enthusiasm!

Developers can now meet the business executives' request for timely access to key information in order to make important business decisions. And, not only did we announce this new feature, but the OSAS development team also announced their commitment to implement dashboard widgets and demonstrated that they've begun the work. It was exciting to see our new utility feature being implemented so quickly.



Mobile Computing was another focus for our sessions. The ability to deploy in a browser and mobile devices is meeting the user's need to become increasingly mobile and flexible. It allows them to react and make time sensitive decisions from anywhere! Browsers are the new cross-platform solution; BUI applications can be deployed for BASIS-authored and non-BASIS systems alike. We presented tips and techniques for employing cascading style sheets and developing for the smaller screen real estate of smartphones and mobile touch devices.



By Gale Robledo
Account Manager

We had great attendance at our sessions and it was good to see old friends and customers, and meet new ones. We also had time for offsite customer visits in the Minneapolis area to have that invaluable face-to-face time.

The OSAS development team is hard at work with all of the exciting features they presented at the conference, and we are in-step with them to ensure they have the tools and support they need to get the job done. Well done, Open Systems, Inc., for another informative conference! ■



By Bruce Gardner
Technical Support
Supervisor

BBj Logs Revisited

Nine years have passed since our first Advantage article spotlighted the details of BBj® logging. BBj has grown immensely during that time with the addition of powerful new features such as data replication, BUI, the Jetty web server, and auditing. BBj logging has necessarily kept pace with the changes, as has the tool used to manage them – the Enterprise Manager (EM). With that, it seems time for a fresh look at how BBj manages the logs and where to find them.

Q: Where are BBj's installation logs located?

A: BBj's installation logs now appear inside the user's home directory, in a directory named **BASIS** as shown below, where **<username>** is the name of the user account used to install BBj.



Windows
%UserProfile%\BASIS



Mac
/Users/<username>/BASIS



Linux
/home/<username>/BASIS

If you encounter installation problems, the BASIS Technical Support team will request the following logs from this BASIS directory:

- install.log
- install.properties

Q: Where can I find BBj's run-time logs?

A: The default location for BBj's run-time logs is **<bbjhome>/log**. BASIS Support will often request all of the logs in this directory. In addition to providing clues about any errors that are occurring, the logs in this directory provide important information about memory usage, important property settings, JVM version, and much more.

Q: How do I send the logs to BASIS Technical Support?

A: Just zip the entire log directory and email it to support@basis.com. The more information you can send, the better. Occasionally, you may see a large **<PID>.hprof** file in your log directory; EM can generate this manually or it can occur on its own if BBjServices encounters memory problems. Ask whether to include this before proceeding. If the attachment is too large to send via email, you may upload the file to our secure server at upload.basis.com.

Q: How do I compress the log files into a .zip file?

A: All modern operating systems have a utility that allows you to compress files or folders.

Windows

- Use the File Explorer to navigate to your **<bbjhome>** directory.
- Right-click the log directory and select Send to > Compressed (zipped) folder.

Mac

- Use Finder to navigate to your **<bbjhome>** directory.
- [Ctrl]+click the log folder and select 'Compress' log.

Linux

- At the shell, navigate to the **<bbjhome>** directory.
- Type the command: **tar cvzf log.tar.gz ./log**

Q: Can I change the location of these log files or any other settings?

A: Yes, you can change the location, as well as such settings as the maximum log size and log file rotation frequency. Occasionally, BASIS Support may ask you to change the debug levels for some of the logs. Make these changes in EM under the Settings module.

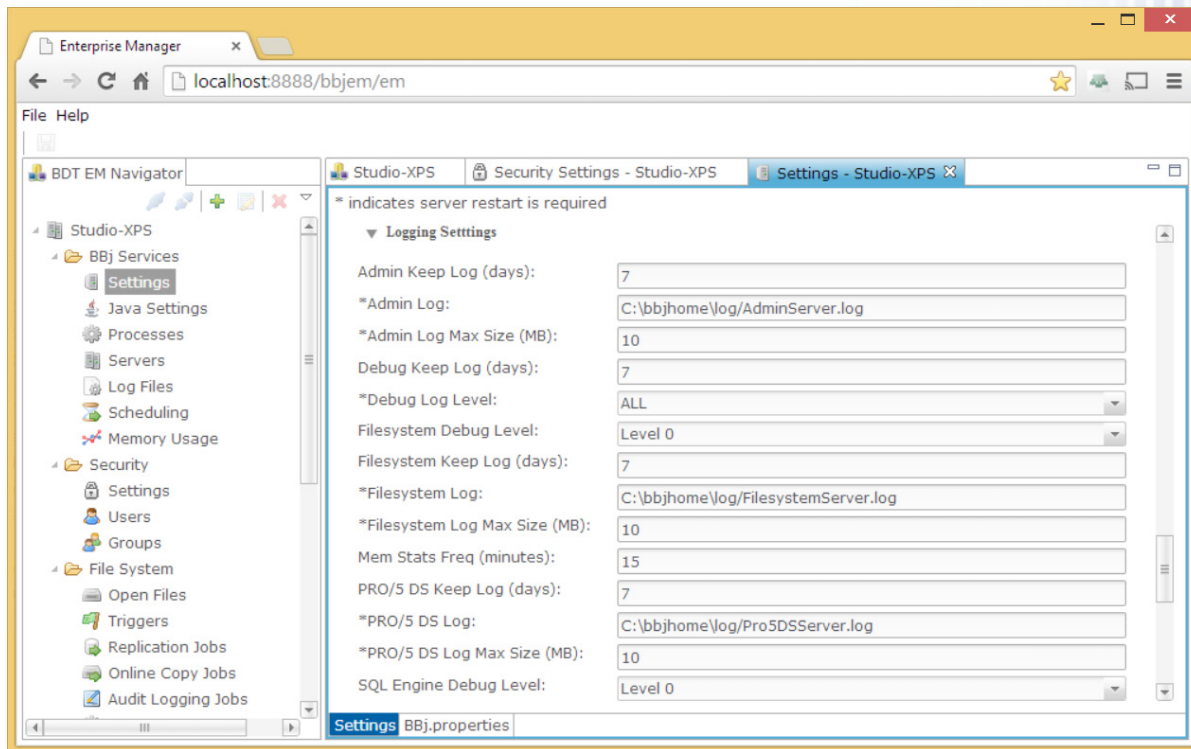


Figure 1. Log settings in BBj's browser EM

Q: How do I review the logs and what am I looking for?

A: You can view various logs directly in the 'Enterprise Manager/Log Files' module. You will notice that separate logs are created for each individual service (see Figure 2).

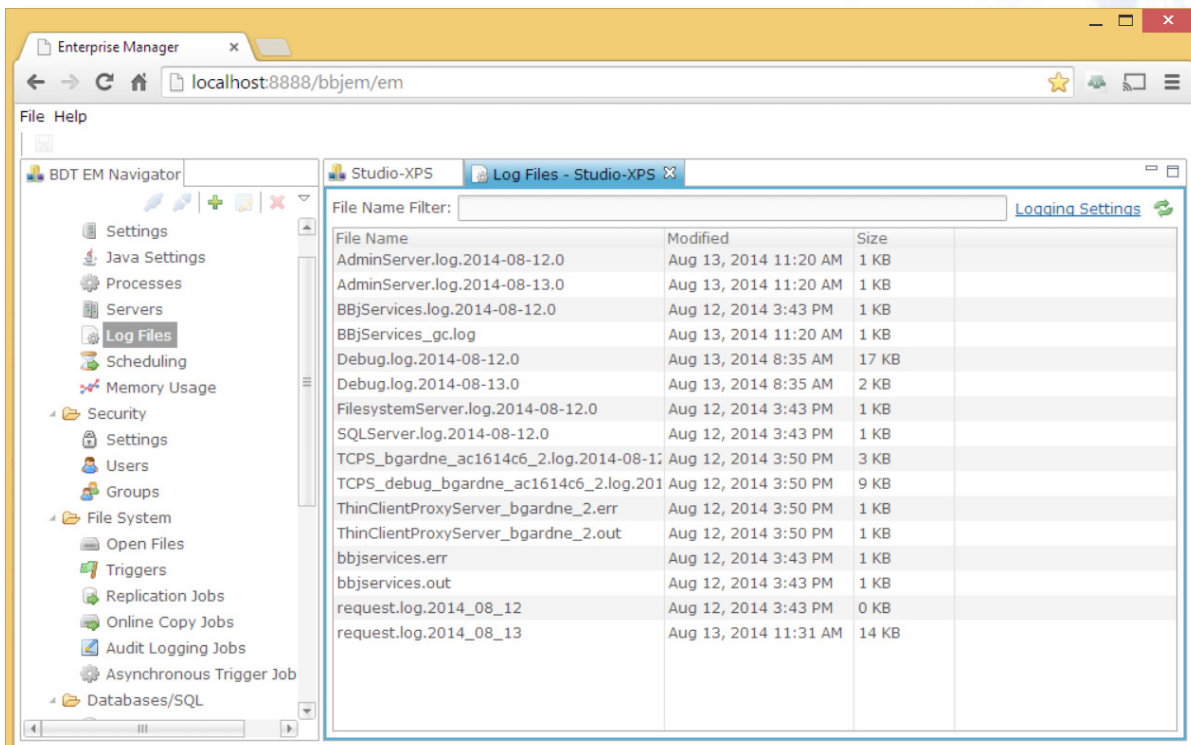


Figure 2. Viewing the BBj logs in the Enterprise Manager Log Files module.

If you're encountering SQL problems, take a look at the `<bbjhome>/log/SQLServer.log.<date>`. Problems with the BBJ PRO/5 Data Server®? Look at the `<bbjhome>/log/BBJPRO5DSServer.log`. The new EM even enables searching for occurrences of text using matches or regular expressions. This is very powerful and has a number of uses. For example, you can quickly find out which programs should be SAVE'd in an updated format simply by filtering for the word 'Upgrading' (see **Figure 3**).

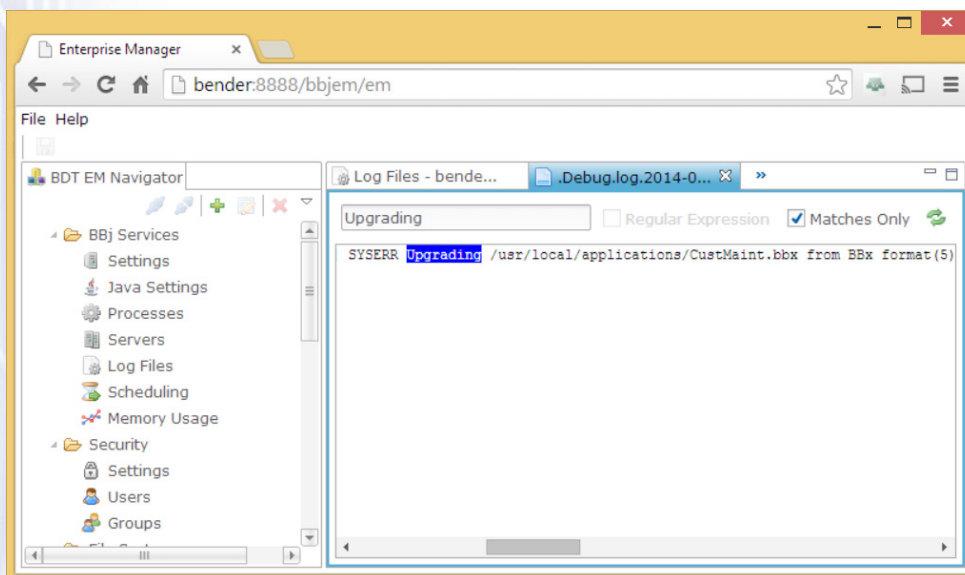


Figure 3. Filtering BBJ log content by keyword

BASIS engineers strive to empower the end user by providing meaningful, human-understandable logging whenever possible. Once you've identified an error that coincides with your BBJ problem, search our website for the error. Our support team is continuously creating and updating Knowledge Base articles, providing solutions to common errors.

Another resource is the BBJ Developer list (subscribe at www.basis.com/discussion-forums). In operation for over ten years, a quick search of the list for an error will often turn up a thread in which someone else has encountered the same error. If a new problem has suddenly cropped up in your deployment, try comparing the current logs to those from previous days. You'll often find the hint you're looking for to resolve your problem.

Q: When Tech Support asks me to perform a "thread dump" or "heap dump," what are these dumps and how do I create them?

A: Thread Dump: This log contains information about the threads and processes currently running and can be a very helpful troubleshooting tool. To generate the thread dump, right-click the server name in EM and select 'Dump JVM Threads' (**Figure 4**). BBJ then writes the thread dump information into the `Debug.log.[date].# log file`.

Heap Dump: This log is useful for analyzing memory-related problems in the Java stack. To generate the heap dump, right-click the server name in EM and select 'Dump Heap' (**Figure 4**). The file usually writes out to the `<bbjhome>/log` directory in the following form: `memoryDump<PID>.hprof`.

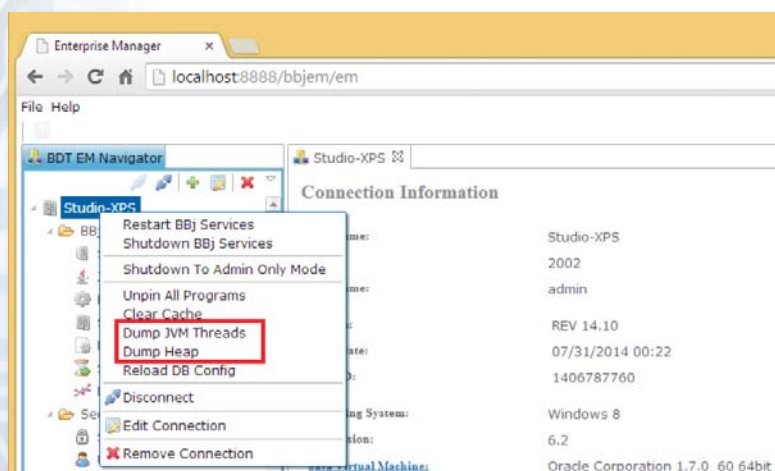


Figure 4. Performing a Thread Dump or Heap Dump from the Enterprise Manager

Q: I am having problems with my Web Start application, which logs should I provide to BASIS Technical Support?

A: Java Web Start is the most popular method of deploying BBJ thin clients. If you encounter problems with your Web Start application, you should send the server-side logs mentioned above and the Java Console output from the client. The Java Console is enabled under the Advanced setting in the Java Control Panel as follows:

1. In the 'Java Control Panel', click the 'Advanced' tab.
2. Expand the 'Java console' option.
3. Select 'Show console' (see **Figure 5**) and click [OK].

When running the Web Start application the next time, a Java Console similar to **Figure 6** will appear. Click [Copy] to copy the contents of the console to the clipboard and then paste the contents into a new document in your favorite text editor.

Summary

Today's BBJ logging capabilities are stronger than ever before, providing a clear picture of the everyday workings of a BBJ deployment. If you are unable to read the tea leaves, the BASIS Support team stands ready to help. Remember, when in doubt, send more files instead of less! ■

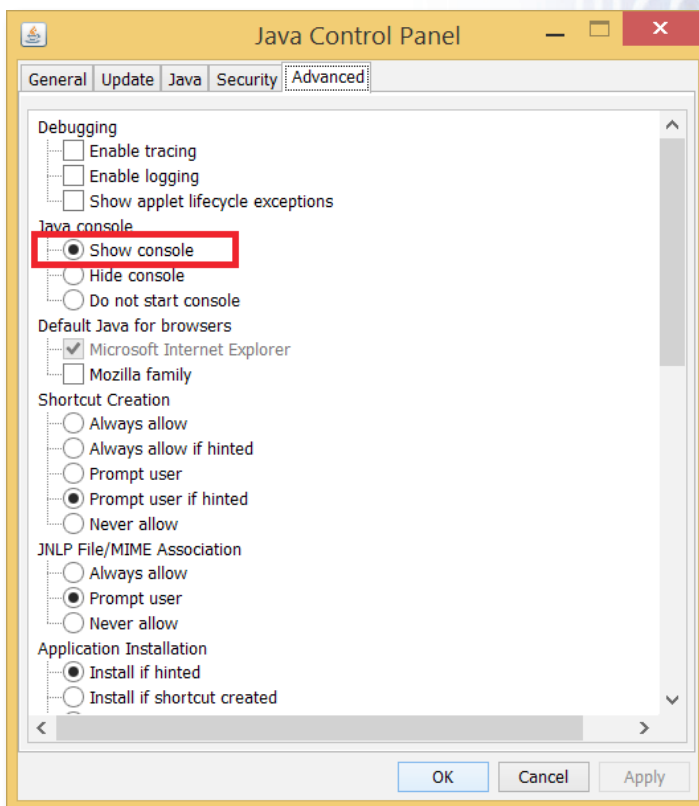


Figure 5. Java Control Panel Advanced Settings where the Java Console is enabled

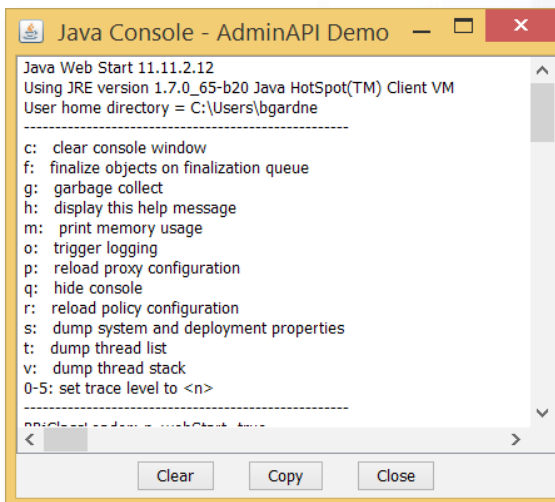


Figure 6. Sample Java Console output

TechCon2015 is Coming!

October 12-14, 2015
Albuquerque Marriott Uptown

TechCon returns to Albuquerque, in the Land of Enchantment and home to BASIS' headquarters. New Mexico is at its finest in October as TechCon convenes on the heels of the Albuquerque International Balloon Fiesta, when the weather is stellar and the trees are in full color.

- Three days of "Java Breaks on steroids"
- Two days of face-to-face classroom training
- Network opportunities with colleagues and BASIS personnel and subject matter experts
- Minimal investment for MAXIMUM gain

Mark your calendar today!



"The content is always excellent, my head reels with the possibilities! What I find more valuable than anything is getting the face time with the engineers. During lunch, before and after classes. It's personal and that's what makes it great!" — Don Goslin, Heilind Electronics, Inc.

"It was really cool to meet the customers face-to-face that I 'talk' to on the lists and answer all their questions in person. They always give me a lot of great ideas."

— Jeff Ash, BASIS Engineer

Arrive a few days early to experience the Albuquerque International Balloon Fiesta, the most photographed event in the world!