

BASIS Advantage

Volume 16 • December 2012

BUMPER ISSUE
BIGGEST EVER!



The Anatomy of a Web App Makeover: A Case Study

Page 8



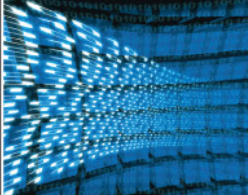
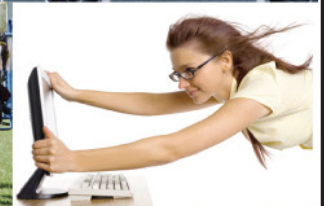
AddonSoftware: Ready, Set, Go Deploy!

Page 14



Barista Caffeinates a CUI App With GUI Sprinkles

Page 28

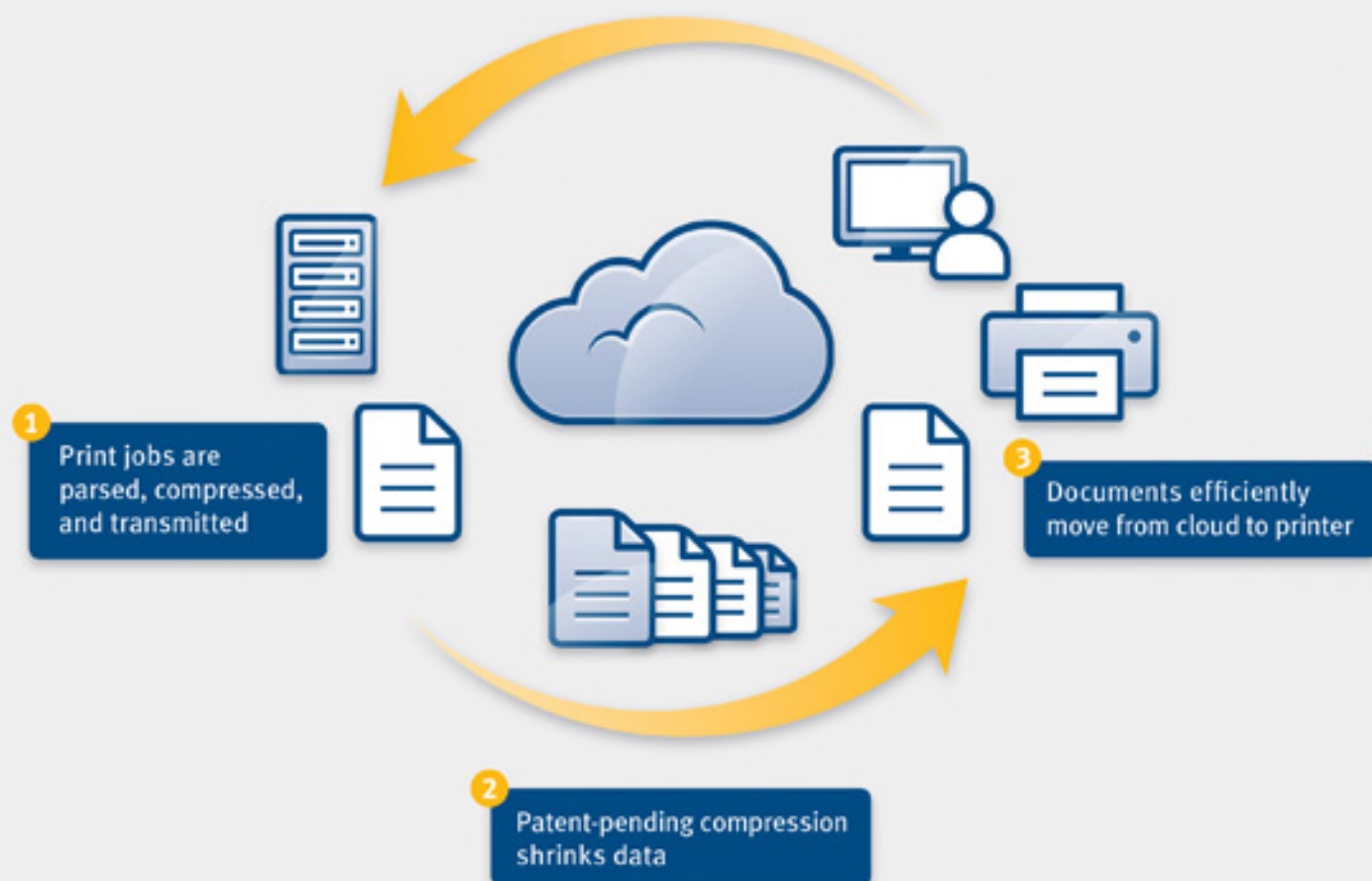


What is an Object?

- The implementation can change over time
- As long as the interface remains unchanged, users of the object are not affected
- Incandescent can be replaced with compact fluorescent

BASIS

CirrusPrint™ Cloud and Networking Printing



Features

- › High performance wide-area network printing
- › Fast, efficient multi-location file transfer
- › Very high compression ratios, up to 98% through patent-pending anti-redundancy techniques, even of already compressed PDF documents
- › One to one, or one to many, document distribution features: one file to many desktops, for example
- › Secure document transmission using SSL
- › Stateless design, so document transmission can occur as remote sites are available
- › Exact replication of print jobs retains features such as duplex and tray control
- › Server-centric configuration via browser interface
- › Reduced bandwidth utilization



BASIS and the TLA* and FLA* IT World

BUI, BYOD, LTE, WORA, RAD, ERP, SFC, BBB, COS, RBE, BCI, GUI...

Six years ago with the release of custom objects, we placed Business BASIC on an equal footing with modern object-oriented programming languages. Three years ago, we prototyped our unique ground-breaking browser user interface (BUI) capability, placing our language at the forefront of computing trends. With the phenomenal adoption of mobile computing by the workforce in the form of “Bring Your Own Device” (BYOD) smartphone, tablet, and ultra light notebook, coupled with blazingly fast LTE wireless speeds, there has never been a greater demand for “Write Once, Run Anywhere” applications and, more specifically, web applications. Our timing couldn’t have been better. In this issue, we offer several articles that demonstrate the reasons for BASIS to be the technology of choice for your next web application.

In parallel to these efforts, we re-examined the reasons for the broad adoption of Business BASIC in the early business computing years and determined that there was a great need for a new set of application building blocks to fuel a fresh round of innovation in the business computing marketplace. As a result, four years ago we acquired

AddonSoftware® along with the forerunner to the Barista® RAD tool, and began the process of modernizing the venerable ERP solution. This issue of *The BASIS International Advantage* coincides with the release of the final module, Shop Floor Control, that completes the third ERP building block bundle; Manufacturing. Together with Accounting and Distribution, plus the myriad of BASIS-supplied new tools and utilities, there has never been a better time to develop or redevelop a vertical market application with the BASIS Building Blocks and the added benefits of BASIS’ unique Commercial Open Source Partnership Program. This issue incorporates articles that highlight several new or newly BUI-ized tools and utilities. From the Resource Bundle Editor, the BASIS Custom Installer to Data Auditing and the new Scheduler in the Enterprise Manager, there is something for everyone in this issue.

We recently assisted a customer in debugging their legacy GUI application code for a ‘lost’ event and were astonished to discover that they executed 53,000 lines of code when navigating through just two lines of a grid – a great testament to our interpreter’s performance while serving as a strong motivator to the customer to refactor their application code to employ

current BBj® grid methods. And so, just like consumers are always going to be on the lookout for a faster notebook or smartphone, we are cognizant to always be on the lookout for ways to make our Business BASIC even faster. We are pleased to say that almost all aspects of BASIS technology benefit from this continued introspection. The interpreters, the database, the utilities, Barista and BUI all perform better and more quickly in the latest releases. The background details on how we achieved these gains are covered in several of the articles in this edition.

BASIS is positioned better than ever to satisfy all of your application development and deployment needs. From our largest 2,000+ user system down to our smallest single-user solutions, from a cloud-based deployment to a desktop deployment, we are ready to partner with you on your next development project.

Come to Las Vegas for TechCon2013 May 13-15 and for training May 16-17 to experience, in person, all that BASIS has to offer...

“Develop Once,
Click-Tap Everywhere!”

See you in Vegas. ■



Nico Spence
Chairman & CEO

* TLA, FLA: Three-Letter Acronyms, Four-Letter Acronyms

Editor in Chief Nico Spence
nspace@basis.com

Editor Susan Darling
sdarling@basis.com

Technical Editors Dr. Kevin King, Nick Decker
kking@basis.com, ndecker@basis.com

Copy Editor Peggy Lewis
plewis@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 Sales 1.800.423.1394
International +1.505.338.4188
www.basis.com info@basis.com

© 2012 by BASIS International Ltd.

Stock photo credits: www.123rf.com

Building Blocks	Partnership	Language/Interpreter	Development Tools	System Administration
 14	 71	 8	 28	 52

This bumper crop of articles spans an all-time high of 100 pages!

Fitting all 33 articles into one issue was as impractical as trying to choose which 10 or 12 articles are the most important to print, as we have done in the past. Each article has tremendous value and is worth reading, so we expanded the table of contents to help you pick out those you want to read first. Find all these articles online plus a few we printed here as samples of what lies ahead! Keeping up with the ever-changing technology around us, we have made *The BASIS International Advantage* more modern, more mobile, and bigger and better than ever. View this ezine everywhere you go — on your desktop, laptop, tablet, smartphone — or print your favorite articles to pass along to others!

Partnership

50 ASCI Partakes RADically From the Barista Cup By Susan Darling



Gain inspiration from this real-world example of how ASCI RADically improved their GUI application with Barista, the BASIS GUI RAD tool. ASCI first introduced improved navigation through Barista's MDI interface with enhanced search, sort, and filter capability; new report output options, tables/forms creation, and role-based security with an audit trail — all built-in features of Barista. Next, ASCI extended the features across all their remaining ASCI modules. Follow their example and RADically change your GUI application with Barista's out-of-the-box features and functions. More than just a RAD tool, Barista's framework delivers building block components integrated into the product. links.basis.com/12asci

71 EMQUE On Cue With BUI Apps That are 'In' By Susan Darling



See the widespread results in action of this pioneer's introduction of BUI apps to the commercial construction community. During the past two years, EMQUE's apps have risen to the top in several notable New York City renovations and have received awards and favorable press, both locally and nationally. Read how they are 'wowing' prospects in the field with new innovations. Consider their strategy, "to grow their apps and market their name to be in position when markets open up and spending increases." Take in their sage advice as you continue or consider beginning your own BUI journey. links.basis.com/12emque

74 A Web Service Sprouts Great Benefits at Bluegrass By Susan Darling



Graft a Web Service onto your app and watch the benefits sprout. A Web Service is still the easiest and best way to communicate, share data, and invoke functionality between disparate systems written in different languages and running on different platforms.

Take a close look at how this BASIS customer sowed their PRO/5 application on an IBM AIX UNIX system with a Windows-based Web Service that brought great cost savings and tremendous increase in productivity. Follow their example and take this first step. links.basis.com/12bluegrass

83 EMS Prescribes BUI to Reduce Healthcare Expenses By Susan Darling



Consider this cure for easily extending legacy PRO/5 data access to desktop and mobile devices. Learn how EMS harvested the 'as is' business logic from PRO/5 into BBj/GUI to deliver the face of the application through an Internet web portal, complete with login security. With minimal training, users can now use a

name or subscriber number to view claim history and verify services to be covered before rendering them, from any browser-compatible device and without adding any JVMs or additional third party programs onto their device. links.basis.com/12ems

Language/Interpreter

8 The Anatomy of a Web App Makeover: A Case Study By Nick Decker



Examine how BASIS rewrote a web page that resulted in a BBj code-based web application that was several times smaller and much easier to write, test, and debug than the original solution — the perfect fit for your next web project. See how BASIS added new features, addressed latency concerns, and built-in form validation. Consolidating all of the underlying code for the download page into BBj greatly simplified testing and the process of optimizing screen layout. For the user, the new download page is faster, easier to navigate, and delivers new features such as localization and real-time translations. links.basis.com/12webapp

22 Going Fast, Faster, Fastest By Adam Hawthorne



Experience maximized performance in BBj programs resulting from significant BBj 12 optimizations to the string handling code and in various verbs and functions with some Java 7 enhancements to the operation of the Just-In-Time (JIT) Compiler. BASIS engineers commit themselves to improving the performance of BBj code so you can concentrate on just writing your application! links.basis.com/12optimize

33 Compressing Apps for Zippy Network Performance By Jason Foutz



Cut the bandwidth required for initially launching a thin client on first install or after an upgrade by over 60% with the latest release of BBj! The secret? Pack200 and Gzip. Jetty, the built-in Web Server that ships with BBj, automatically compresses jars using Pack200. BBj's Web Server also uses Gzip compression to speed up both BUI and static resources served from the htdocs directory, providing the biggest benefit when serving text files. links.basis.com/12zipapps

42 Looks Better, Runs Faster By Nick Decker



Follow these simple suggestions for optimizing your application's images to achieve quicker display times, make the most of limited bandwidth, while delivering quality output. Learn about choosing the best tool for the job to ensure that images render in the highest quality possible.

Fortunately, optimizing images is fairly easy and quick, especially with some of the advanced compression tools available. You can now accomplish this task yourself instead of delegating it to a dedicated graphic artist. BUI applications look great out-of-the-box on the new Retina and pixel-doubled displays without the need to resort to custom code! links.basis.com/12image

DBMS

26 Turn On Data Auditing By Jeff Ash



Activate the new Change Audit Logging that makes the process of building an audit trail fast and easy to implement for changes to all BASIS file types. Audit logging data resides in a BBj ESQSQL database so the data is accessible from either the Audit Log Viewer in the Enterprise Manager, or from

querying the audit database directly using SQL. Configure user access permissions to the audit database easily and in the same way as on any other BBj database. Using iReport or BBjJasper, administrators can create more robust, limited, and/or customized reporting for others to view in an external application without the need to grant them access to the Enterprise Manager. links.basis.com/12auditlog

40 Quickly Fix Slow SQL By Jeff Ash



Cut down on the time necessary to analyze your data tables. The improved Query Analysis interface makes it easier to locate potential areas for improvement in SQL query performance by better indexing the tables based on the user's query usage of those tables. Find out how one large database that previously took 11 hours to

analyze in BBJ version 11.11 now performs the analysis in 3 hours using BBJ version 12. Dramatic performance improvements are just a click away! links.basis.com/12performance

54 Built-in SQL Access to BASIS Keyed and CSV Files By Robert Del Prete



Access any BASIS or CSV formatted data file regardless of the presence of a data dictionary with new built-in stored procedures. These new stored procedures are always available and can be called from a connection to any available BBJ database. If there is no database defined within BBJ Services, use the system database to execute the stored procedure. Leverage this new feature in many ways, such as supplying new access to data by end users. Being able to filter and sort the data using SQL syntax also provides additional flexibility for the distribution and presentation of that data.

links.basis.com/12sprocs

86 BASIS SQL Gets Even Better By Jeff Ash



Expand your SQL functionality with new SQL scripting, types, verbs, functions and views. Easily execute scripts inside the Enterprise Manager. The addition of ENUM, REPLACE, DATEDIFF and Full Featured Views gives BBJ developers several more valuable tools for developing database applications. With continued input from

BASIS customers offering ideas for improving our products, you can count on more improvements and new features becoming available to the BASIS toolset.

links.basis.com/12sql

Development Tools

18 The Magical Reusable Dialog Wizard By Ralph Lance



Be amazed with the new Dialog Wizard, the modern and powerful alternative to AppBuilder. Generate fully object-oriented BBJ programs from a resource file, automatically generating the callbacks and the method stubs while preserving your custom editing of the resultant BBJ code. You can run a second or third pass

of the code generator to leverage any changes you may have made to your resource file. See just how quick it is to go from a resource file to a running program. Beyond magic.

links.basis.com/12dialogwizard

46 REST Easy - End Your WSDL Struggles By Jason Foutz



Stay afloat in the alphabet soup of UDDI, WSDL, and SOAP, and eliminate much of the formality imposed by a WSDL-style Web Service. Employ the spectrum of Internet infrastructure with REST-based Web Services. REST is stateless and leverages HTTP, simplifying the communication to the server. Choose this powerful tool

for deploying Web Services without the risk of drowning in alphabet soup.

links.basis.com/12rest

58 Babbling With the New Bundle of Joy By Brian Hipple

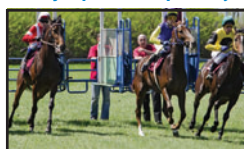


Meet BASIS' new Resource Bundle Editor. Easily separate displayed string literals from the application code, whether or not your app babbles many languages. When separating these string literals from the code, maintenance is so easy that non-programmers can even change the text. Better yet, a single update of the string

propagates the change throughout the entire application regardless of how many times it appears. Welcome this new bundle and add it to your family of tools.

links.basis.com/12bundle

63 'Git'dy Up, Developers By Shaun Haney and Christine Hawkins



Jump out of the starting gate and upgrade your customized AddonSoftware app with Git, an upgrade intelligence tool that makes upgrading customized packages easier than ever before while preserving all of your custom code. Don't stay "stuck in time" without the possibility of an application upgrade. With

Barista and Git, developers can now modify both the user interface as well as other code, and still realize the benefits of an upgrade. Get Git and Git'dy on up.

links.basis.com/12git

79 Customizable Mobile Report Viewer By Brian Hipple



Perfectly display interactive reports in a browser on your smartphone, tablet, and other mobile device with the all new BBjJasperViewer utility, written in pure BBJ code and released in BBJ 12. Now you can run BBJ code in BUI as it translates code into the browser-rendered HTML, JavaScript, and CSS. The updated

user interface includes a new child window that houses the various tool buttons that manipulate the report. Preview the BBJ 13 feature that makes it possible to use and display JasperReports in applications without using SQL or SPROC's. Realize the power of BASIS' BBjJasper application building block utility today!

links.basis.com/12reportviewer

82 BASIS IDE in Java 7th Heaven By Mike Phelps



Jump ahead and update to Java 1.7 before the Oracle clock runs out. The BASIS IDE now supports Java 1.7. BASIS has made this IDE upgrade so you can avoid any last minute crises due to Java 1.6's looming end-of-life.

links.basis.com/12ide

88 Easily Install Your Apps With the BCI By Brian Hipple



Hand off the simpler time-consuming tasks so you can focus on those more complex. It's time to work smarter rather than harder. The installation process is one you may never have to do manually again.

The BASIS Custom Installer (BCI) makes installation more simple than ever, with several new custom options. Now you can configure the BCI to invoke specific commands under specific conditions, install certain files, and run your choice of BASIS installation wizards in the manner you choose. Imagine being able to download, install, configure, and run your app without any user interaction! links.basis.com/12custominstall

94 BBJ Documentation is as Easy as JavaDocs By Ralph Lance



Imagine creating API documentation automatically and directly from your BBJ object-oriented application code. The new BBJToJavadoc utility, a Javadoc documentation generator, does just that.

Learn how you can easily embed documentation comments in your BBJ programs to generate your own documentation in HTML Javadoc format. BBJ custom classes can take advantage of some of the more advanced documentation capabilities for easy navigation via hyperlinks among your classes. Because the Javadoc engine from the Java JDK toolset is used in the process, any valid document tag can be embedded in the doc comments. links.basis.com/12docs

System Administration

52 BUI, GUI Everywhere By Shaun Haney



Give your app worldwide equal opportunity access with distributed BUI apps. BASIS' replication feature and geo-aware DNS (Domain Name System) addresses deliver quick performance anywhere in the world. Explore how to combine Amazon's geo-aware DNS addresses with BBj's File/Directory/Database

Replication to run your app equally well and transparently from any location on the globe. Follow along this detailed example to learn about the tools, the architecture, and how it all works. links.basis.com/12gui

66 Platform-Independent Task Scheduler *By Jeff Ash and Nick Decker*



Put power and flexibility at your fingertips with the new platform-independent scheduler. Replace your Windows Task Scheduler or UNIX cron jobs with BBJ Tasks and Task Groups for scheduling tasks that can now even interact with BBJ Services. Walk through this example and learn how to manage BBJ

backups and other business processes at regular intervals. Save time, and save the hassle of using one or more third party schedulers by using this built-in easy-to-configure and manage tool. links.basis.com/12scheduler

90 Are You Prepared for Cloud Failure? *By Shaun Haney*

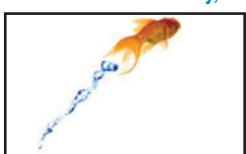


Depend with confidence on the cloud for your business continuity, applying these important tips and safeguards. Redundant copies of data over multiple regions will ensure that in most any outage, your enterprise can continue running, unscathed. Follow the tried and true example that

BASIS put in place to protect against the potential loss of data and money that would result from a cloud outage. Learn the vital strategy for quick recovery in the event of such an outage. links.basis.com/12cloud

Building Blocks

14 AddonSoftware: Ready, Set, Go Deploy! *By Christine Hawkins*



Build or supplement your vertical market with this suite of Accounting, Distribution, and Manufacturing application building blocks and its newly ported Shop Floor Control module. This milestone marks the beginning of a host of modernizations and improvements now possible with AddonSoftware's

new foundation in BBJ and Barista. Why not let BASIS take care of providing the ERP building blocks of your vertical market application? Try it today!

links.basis.com/12addon

28 Barista Caffeinates a CUI App With GUI Sprinkles *By Ralph Lance*



Integrate fully-functional graphical components with the least amount of effort using the Barista Application Framework. Instantly begin benefiting from the myriad of built-in features that Barista brings to your application. Follow the example of how to add a GUI lookup to a simple BBJ character-based application. Discover the many variations – sort, filter, select, output, etc. – available to sprinkle into the CUI app. Ease into graphical user interfaces with minimal effort and disruption to existing code by leveraging the out-of-the-box functionality that Barista provides. links.basis.com/12barista

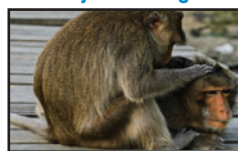
34 DocOut Easily Modernizes BBx Report *By Ralph Lance*



DocOut is a document output subsystem component of the Barista Application Framework RAD tool. Let DocOut give you all the output choices for your reports that you need for a modern app – print preview, PDF, XLS, CSV, XML, Google Docs, fax, email, and archiving. Now you can leverage these

choices from new or existing PRO/5, Visual PRO/5, or BBJ code without having to use the entire framework, and with very little programming effort! See how it's done and try out the sample code. links.basis.com/12docout

76 New Ways to Debug in Barista *By Ralph Lance*



Simplify your troubleshooting efforts in your Barista and custom callpoint code with many new debugging features. For example, debug and dot-step through the code by pressing the [ESCAPE] key to interrupt the process. Or view the dump to see the contents of the workspace, and start and

stop tracing. You can even view the namespaces and the contents of the namespace variables. Using Barista is better than ever, so add it to your toolset and enjoy the easy and efficient way it can help you be more productive in the support of your customers.

links.basis.com/12debug

Develop Once, Click-Tap Everywhere

BASIS TechCon2013 in Las Vegas, NV

May 13-15 – Conference

- New Eclipse IDE
- Enhanced Data Dictionary
- Mobile Enterprise Manager
- Faster, slimmer, better BBJ and BUI
- Three-Tier Architecture Dos and Don'ts
- Building Blocks - New and Revamped

May 16-17 – Training

- CSS for Web Apps
- CUI to GUI Best Practices
- Eclipse IDE
- DBMS Enhancements
- ERP Building Blocks
- BASIS' Amazon

Register today links.basis.com/tcreg

Columns

39 BASIS Unveils New Training Format *By Amer Child*



Check out the new web-based training classes that incorporate pre-recorded content with live instructor interaction. Pre-recording the core instruction ensures students receive consistent delivery of all the important points needed to understand the key concepts for each class.

Find out why the feedback on our first few training courses in this new style was positive with both students and trainers. Try out this win-win approach.

links.basis.com/12training

48 BASIS Survived Amazon Outage *By Dr. Kevin W. King*



Keep your website and other cloud applications running 100% with little to no risk of a calamity bringing your business to a screeching halt. As BASIS has gone before you to build in safeguards to avoid such a catastrophe, read how they avoided the infamous Amazon outage that took out so many larger and seemingly less wise companies. Follow the leader is more than a child's game, it can be a very valuable business model. Learn from BASIS so you can benefit from all the power and convenience and cost savings of the cloud while removing the risk along the way. links.basis.com/12survived

62 An Insider Look at BASIS Testing *By Aaron Wantuck*



Feel confident in the extensive tests that BASIS runs before releasing a product. Inquiring minds can now be satisfied with the details of the test suite in place that rigorously runs through each build. Discover the two parts of the suite – Junit and testbed – and better

understand the roles they play. Thorough testing ensures that we can accomplish our goals while continuing to meet yours. links.basis.com/12testing

68 Continuous Innovation at BASIS *By Mike Phelps*



Look behind the scenes at what makes BASIS a stronger, more successful company...something that indirectly benefits everyone who uses BASIS technology. Once considered science fiction a few years ago, their new build system is completely

offsite "in the cloud." Along with the build process, testing and delivery is 100% cloud-based and is fast, easy to operate, almost infinitely scalable, and (we believe) disaster-proof.

links.basis.com/12innovation

92 Java Breaks Deliver *By Paul Yeomans*



Discover how the bimonthly Java Break sessions solved a key information delivery challenge; to reach the BASIS community with relevant information more current and frequent than a bi-annual TechCon and at a global location. Java Breaks to the rescue!

links.basis.com/12javabreak

98 What's Needed to Run BBj? *By Bruce Gardner*



Ask anyone and the answer is a resounding "well, it depends." With so many possible variables in a BBj application deployment, mostly because of the variability of the application itself, it really is impossible to provide a specific answer. After all, one size does *not* fit all. However, all of the tools

necessary to assess the requirements for your deployment are at your fingertips. Examine these tools for a full understanding of the dynamic factors involved in all applications and their deployments, and determine those requirements yourself.

links.basis.com/12trz

Cloud Hosting for BBj Applications

- High availability
- High performance
- Optimized for BBj
- Fixed monthly costs



- VMware clusters
- Migration assistance
- Replication for Disaster Recovery
- Dedicated clouds available

www.bbjcloud.com

Customer Maintenance

File Help

Customer Account History

Number: 000001 Status: Active

Name: Cloud Sales

Company: BBjCloud.com

Address: 111 Commerce St
Lake Mary, FL 32746

Preferred Contact

Email: ☒ sales@bbjcloud.com

Phone: ☐ (800) 840-8649

Fax: ☐ () -

☒ Receive Product Information

GWT Browser Firefox

Browser
User Interface
(BUI)

Serial Number History 002157

Previous View Link Print License Select a reason for reset

Serial Number: BBX615871

Item: BASALL166CUREXP

Description: BASIS Licence, Windows, Expiring Product

Date	Invoice	Activity	Revision	Users	Key/Authorization
18 Feb 2009	0155934	EXC	8.xx	2	0207558711
18 Feb 2009	0000000	UNC	9.xx	2	0207558711
14 Jan 2010	0000000	UNC	10.x	2	0207558711
15 Feb 2010	0163360	RINw	10.x	2	0207558711
4 Mar 2011	0170594	RINw	11.x	2	0207558711

Thin Client

Call us at 855-vCLOUD-8 (855-825-6838) or 407-574-6986
Email info@bbjcloud.com

BBjCloud – A BASIS Authorized Distributor



The Anatomy of a Web App Makeover

The BASIS Product Suite Download page, written years ago with a mixture of Perl, HTML, JavaScript, and SQL, was in great need of a makeover to address a growing list of enhancement requests from our community. As a testament to the complexity of this download page, its Perl code alone relied on a long list of external libraries to integrate critical functions such as CGI, database, email, FTP, SOAP, date/time manipulation, SSL and cryptography, and MIME encoding integration. With code that spread out over multiple files and various languages, this page became increasingly difficult to maintain and add the user-desired features. The time had come to rewrite it in a simpler, consolidated, and more easily maintainable language that was capable of doing everything the previous system could do and more to accommodate the upcoming improvements. Our tool of choice – BBJ®, of course! With BBJ's built-in BUI functionality, it was an easy decision.

New Features

The new BUI (browser user interface) download page delivers a nice list of new features available in its first release:

- Localization for five different languages
- Locale auto-detection with the option to select a language at any time for real-time translation (see **Figure 1**)
- Build timestamp display
- Reduced amount of required contact information
- Dynamic build retrieval from an [Amazon S3](#) bucket

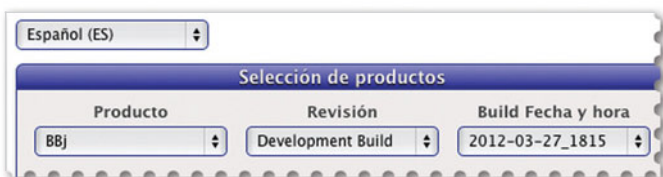


Figure 1. An example of the locale auto-detection



By Nick Decker
Engineering
Supervisor

The Latency Test

Concerned about performance of the download page under high latency conditions, we launched a copy of our BBJ production server's [Amazon EC2](#) instance in Ireland. Surely, a distance of about 4,700 miles ought to be significant enough to introduce some network delays from the BBJ interpreter in Ireland to our browsers in Albuquerque! The average ping time from Albuquerque to the West Coast production server was about 45ms, while the ping time to Ireland was more than five times that amount – an average of 245ms!

Latency can have a dramatic impact on any program where the client and server must pass information back and forth. In the case of the product download page, BBJ was running in the BUI paradigm, but the same would hold true for a thin client deployment of the code. Even though the new download page has fewer fields to complete compared to the old page, the program running on the server still needs access to the information in all of those fields. In order to ensure that the user fills in the required fields appropriately (**Figure 2**), and to make use of the information later in the process to check export compliance, the BBJ program needs to retrieve the contents of the 17 form elements from the client's browser.

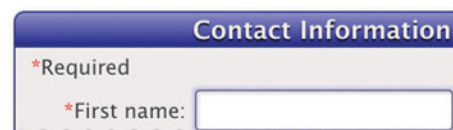


Figure 2. An example of one of the 17 required fields on the form

Making that many round trips to the client can be really time-consuming, especially as network latency increases. Our initial round of testing revealed what we had expected; validating the contents of controls on the user's browser was very time consuming.

If only there was some way to avoid asking the client 17 times for information regarding the status and contents of the controls on the form. Wouldn't it be nice if the interpreter could magically harvest all of the information at once?

Enter an Old Friend - Form Validation

It turns out that BBJ has had a related paradigm already in place for a traditional thin client deployment – BBJFormValidation (see *Input Validation - Veto Power* at links.basis.com/05validation). The concept of form validation has been around for as long as the Internet itself. Because it becomes expensive to repeatedly ask the client for information regarding the contents of controls in response to user changes, web pages wait until the user completes entering all the changes before validating the fields. Once the user presses the [Submit] button, the program proceeds to validate all of the fields on the form. Beginning in version 12, BBJ adds the same form validation that was previously available in the thin client to the BUI paradigm.

Adding form validation is the first of two improvements that we implemented to solve the download page's latency problem. While form validation allows the programmer to lock the form preventing the user from making any further changes until the validation has completed, it does not do anything to reduce the number of round trip communications between the server and the client. We addressed this second part of the problem with a dramatic improvement to the BBJFormValidation event itself; it now carries a payload of information with the state and contents of the controls on the form. This means that the programmer's validation routine no longer has to ask the client for the text in a BBJEditBox, or whether a BBJCheckBox was selected or not. Instead of asking the client, which causes network delays, the program simply retrieves the desired information from the form validation event itself on the server.

Retrieving information from the [Form Validation Event](#) was so useful and revolutionary, that BASIS made similar changes to many other events unrelated to form validation. For example, we augmented all of the BBJList events with information about the control's state. Previously, the BBJ program registered for the BBJListSelectEvent so that it could react to any changes in the selection of a BBJList control. When the user selected an item in the list, this action notified the program and would then execute the callback routine or method associated with that event. In most cases, however, the program would immediately turn around and ask the control for the item in the list that the user selected. After all, if the program cared that the user selected an item in the list, chances are pretty good that it also cared which item in the list the user selected. Therefore, whenever the user selected an item in the list, the event would result in another round trip question/answer from the server to the client in order to find out the new selection in the list control.

To eliminate that round trip, BBJ 12 augments the BBJList events with three new events:

1. `getSelectedIndex()` - returns the currently selected item in the list
2. `getSelectedIndices()` - returns a vector of all currently selected items in the list
3. `getSelectedItem()` - returns the text of the currently selected item in the list

These new events allow the BBJ program to respond to a selection event in the list control and have all of the information it will need ahead of time from the event itself. BASIS also added these same improvements to other events, such as BBJLostFocusEvent and BBJEditModifyEvent. It is worth mentioning once more that these new methods are

available in both SYSGUI and BUI, so they will improve the speed of many of your existing GUI applications. **Figure 3** shows form validation in action.

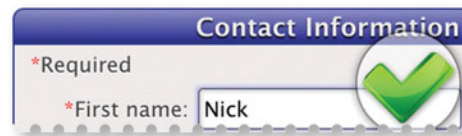


Figure 3. Form validation makes it easy to ensure that the user provides required information

Why BBJ BUI Made Life Easier

Consolidating all of the underlying code for the download page into BBJ greatly simplified things. Some of the more difficult aspects of the old download page, such as the export compliance check, were much easier to implement in BBJ and required far less code. The old page built a SOAP message from the ground up, creating headers, adding data to the message body, and using specially coded routines to handle more complex data types. In contrast, the new BBJ version of the download page utilizes a Web Service to accomplish the same task. The code is simple and straightforward; add a couple of jars to your classpath in Enterprise Manager (EM), instantiate the Web Service client, put some properties in a HashMap, and execute the `runTransaction()` method. The BBJ code (**Figure 4**) is succinct and provides a high level of readability and maintainability.

```
method protected BBJNumber checkExportCompliance()
declare java.util.HashMap request!
declare java.util.HashMap reply!

REM Create a new HashMap
request! = new java.util.HashMap()

rem Add all of the user information
request!.put("billTo_firstName", #savedStateFirstName$)
request!.put("billTo_lastName", #savedStateLastName$)
...

rem Run the transaction
reply! = Client.runTransaction(request!, props!)
```

Figure 4. Excerpt of the export compliance BBJ code

Several other routines were also significantly easier in BBJ compared to the previous version including validation, user notification [LightBox](#), and cookie management. One of the best simplifications to the new program was to access BASIS data files and databases natively. The old version, which relied on Perl for the backend CGI language, used the DBI ([database independent interface](#)) library for database access. This required the `DBD::JDBC` module that works in conjunction with a server written in Java to provide a DBI front end to a JDBC driver, acting as a bridge between Perl and a BASIS JDBC database. The end result is that we were able to eliminate more libraries, remove the dependency on required run-time processes, improve compatibility, and speed up data access in the process.

Having all of the code in a single language and in a single program not only simplifies the process, but makes the coding/debugging/maintaining much easier as well. The old version retrieved values from the browser via JavaScript, validated

them; sent them back to the Perl CGI program on the server, which manipulated them in order to pass them on to the export compliance services in a SOAP message, then inserted them into a database. Not only do different languages like JavaScript, Perl, and SQL each have a vastly different syntax, but they also have different data types that can complicate the passing of variables from a program in one language to another program in a different language. Even comparing data in JavaScript and Perl is very different; JavaScript uses the `!=` operator to test for inequality, while Perl only uses `!=` when comparing numbers and instead uses the `ne` comparison operator when the value is a string. Having all of the code and data in a single language reduces the demands and requirements for the programmer and eliminates the need to keep track of all of the special rules for various languages as illustrated in **Figure 5**. The end result is a program that is much more robust, less prone to breakage, and far easier to debug.



Figure 5. Consolidating everything into a single language - BBj

Testing was Easier Too!

Because we developed the new download page inside the BASIS IDE, testing this page was also very easy. Simply tapping the [F5] key or clicking the [BB Execute] button instantly launched the program in thin client mode. Adding a BUI definition for the app in EM only took a minute, so testing it in BUI mode in a browser was also straightforward. EM provides a link to the BUI app so sending off a quick email to the BASIS QA department with the link greatly facilitated testing efforts. Because a BBj installation includes a fully-configured Web Server that is available when starting BBj Services, the QA department could also check out the BBj program and resource for the download page from our SVN source code repository and then host and test the new page locally.

By contrast, testing the old download page was extraordinarily difficult. We could only run the old page on a system pre-configured with Apache Web Server, Perl interpreter, dozens of external libraries and modules built from source code, runtime Perl/JDBC bridge server, and more. Because setting up and maintaining this server was so extensive and time consuming, it was just impractical and extremely expensive to have more than one development machine.

Comparing the Old and New Download Pages

One goal of BUI-izing the product download page was to reduce the size and complexity of the form. Comparing the old and the new pages side-by-side in **Figure 6** shows our success; the new

 A screenshot of the old download page. It features a 'Available Releases' section with a list of revisions (11.12, 10.04, 9.12, 8.31, Development Builds) and a caterpillar illustration. Below is 'Available Packages' showing various software bundles. The 'User Information' section has multiple text fields for first name, last name, company, address, city, state, postal code, and country. There is also a section for 'Planned or current use of BASIS products' and a 'Download Product' button at the bottom.

 A screenshot of the new download page. It is more streamlined, starting with a language selector (English (EN)). The 'Select Product' section has dropdowns for Product (BBj), Revision (Development Build), and Build Timestamp (2012-03-24_1116). The 'Select Package' section has radio buttons for different software bundles. The 'Contact Information' section has required fields for first name, last name, company, email, address, city, state, postal code, and country. There is a checkbox for 'Yes, send me periodic emails...' and a 'Download Terms' section with checkboxes for agreement to terms and export controls. A large 'Download' button is at the bottom.

Figure 6. Comparing the old (left) and new (right) download page

page is smaller, more compact, and eliminates unnecessary data such as “Fax Number” and unwieldy supporting controls like the “US/Canada” and “Other” radio buttons and the various “If Other” text boxes. BUI’s integration of Cascading Style Sheets also assists with making the form more visually attractive while extending the BASIS website theme, including the same fonts, colors, and gradients, where appropriate.

New Features

One new feature of the BASIS Product Suite Download page is localization into five different languages. While the BUI program uses the BBTranslator Utility to translate the controls in real time, BASIS used the new BASIS Resource Bundle Editor (links.basis.com/rbe) to create and maintain all of the text in specialized resource bundles. When the program starts, it first checks to see if the URL specifies a locale for the BUI application. If there is no locale in the URL, the program checks for a past locale setting that it saved in a cookie. If that is not available either, then the program retrieves the locale from the client’s machine.

Once the program has a valid locale setting, it calls a method to translate all of the controls. If the locale is not one of the five supported languages, then it uses the default language, kept in sync with English by the BASIS Resource Bundle Editor. The user may also change the language at any time by selecting a pre-populated option from the BBjListButton. The callback routine for this list selection event retrieves the selected locale from the BBjListSelectEvent to avoid a trip to the client, then calls the method to translate the controls using the new locale. **Figure 7** shows a portion of the page translated into Italian.

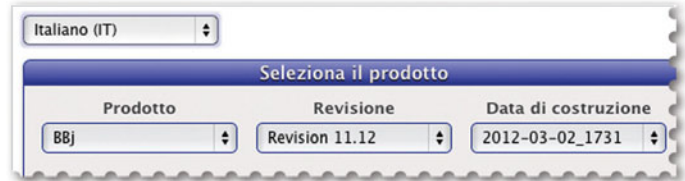


Figure 7. The download page translated into Italian

BUI and GUI Mortgage Demo Optimizations

To illustrate how you might modify your GUI or BUI application to take advantage of the new form-validation’s payload, consider how we applied the changes to the BUI Mortgage demo.

The original version of the code calculated the payment information when the user pressed the [Calculate] button. The shift to form validation does not change much in this part of the code, just the event type for which the button is registered. Instead of reacting to the ON_BUTTON_PUSH event, we now trigger form validation from the button by registering for the ON_FORM_VALIDATION event as shown in **Figure 8**.

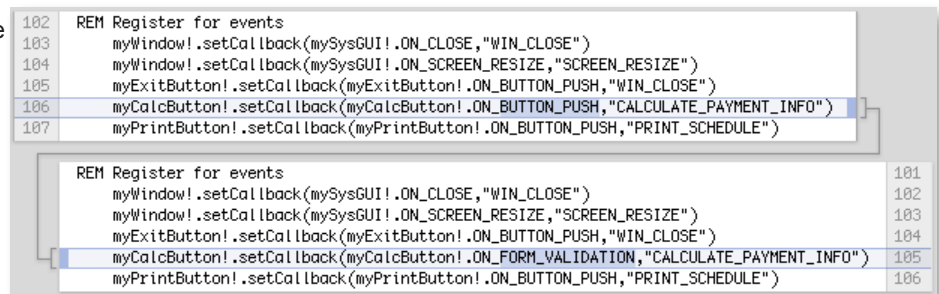


Figure 8. Changing the calculation button’s callback to trigger a Form Validation Event

The next change took place in the subroutine that executes when the user presses the [Calculate] button. To start with, we retrieved the Form Validation Event into an object, highlighted in green in **Figure 9**. Because the event object contains all of the information about all of the controls on the form, the next few changes deal with getting the contents of the various controls.

This is the most important part of the change since it is where we eliminate all of the latency overhead of making round trips asking the client for information. This change is pretty simple, too. Instead of getting the value of the myPrincipal! InputN control directly, we modified the code shown in **Figure 9** to get the value from the form validation event by referencing the Principal! InputN control.

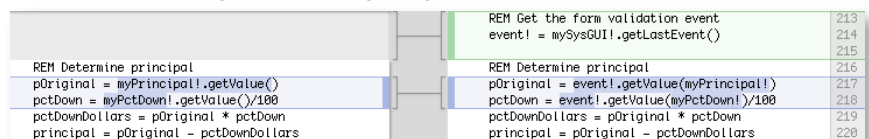


Figure 9. Retrieving a field value from the validation event

That sort of change continues on for the remainder of the input controls. Once the routine has processed all of the input values, it updates the screen with the results. One last change is necessary, as form validation requires the program to accept or deny the validation event. This is due to form validation locking the top-level window from user changes while the program processes the form. In order for BBj to unlock the window and allow additional changes to the form, the program must call the accept() method as shown in **Figure 10**.

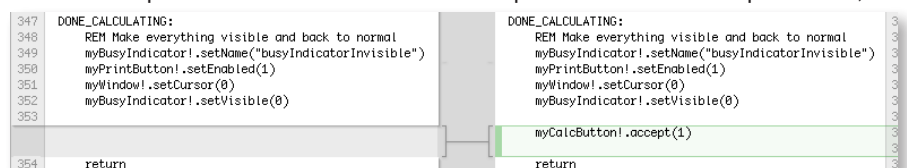


Figure 10. Calling the accept() method to complete form validation

After making the changes in the program from querying the controls directly to using form validation, we ran comparison tests to

a remote server to see the effect. Modifying the program to use form validation turned out to be a winner, making the program more than 11 times faster when retrieving the contents of the controls. That speed improvement resulted from a test using a fairly low-latency connection with a server that was approximately 1,000 miles away with an average of 45ms ping times. In higher latency scenarios, the speed improvement was even more significant.

Optimizing Layout - Screen Detection for Mobile Browsers

Initially, we embedded the new download page in an `IFrame` on the BASIS website, which we built with a Drupal-based CMS system. The main page, shown inside a desktop browser in **Figure 11**, contains the menuing system as well as the left and right navigation bars. In order to maximize the space available for the BUI application when running on a mobile device, we took advantage of pre-written code available from detectmobilebrowsers.com. It offers code in more than 15 languages to detect if the web page is being loaded into a mobile browser or not. We inserted the PHP code into our Drupal page and redirected the client to the BUI-only version of the app if viewing the page in a mobile browser. This maximizes the amount of space for the application on smaller devices such as tablets and smartphones.

Although it may seem odd at first to download BBJ to a phone or tablet, many mobile devices and operating systems offer the ability to save downloaded files

to the cloud via applications like Dropbox. This capability allows you to download BBJ from any Internet-connected device with a browser and save it to a single location in the cloud for subsequent access by numerous machines.

Mission Complete - Optimizations in Place!

Rewriting the BBJ Download page was a smashing success by any measurement. The resultant new BBJ code base was several times smaller, easier to write, test, and debug than the old multi-language code base. We eliminated several other languages, libraries, runtime servers, machine dependencies, and setup and configuration complexities. The new download page is faster, easier to navigate and use, and includes new features such as localization and real-time translations. Best of all, we accomplished all of this with our own toolset, proving that that BASIS' BUI technology is a perfect fit for your next web development project. ■

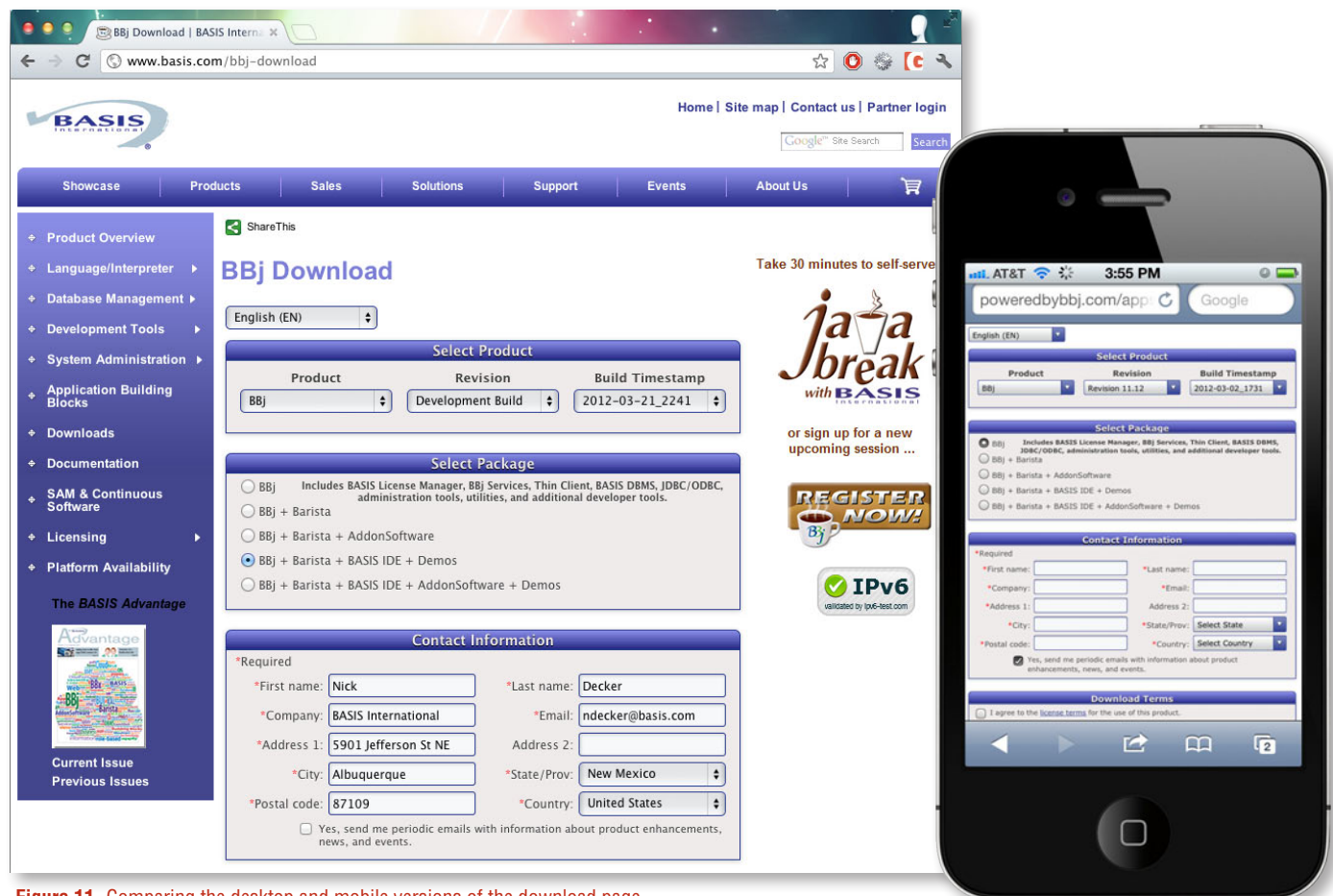


Figure 11. Comparing the desktop and mobile versions of the download page



- Read the *BASIS Advantage* article *Input Validation - Veto Power* at links.basis.com/05validation
- Learn more about the BBJ methods and events mentioned in this article in the online documentation at links.basis.com/basishelp
- Optimizations do not end here - read more in this issue
 - *Looks Better, Runs Faster* at links.basis.com/12image
 - *GUI, BUI Everywhere* at links.basis.com/12gui
- The BUI Mortgage source code is installed with BBJ as part of the demos package

Addon Cloud

AddonSoftware® Cloud Services – Addon Cloud – is a new and simpler way of deploying and upgrading your enterprise resource planning (ERP) software

ERP simpler than ever before!

Significant cost savings – Everything you need is included in the monthly fee

Performance and reliability – Infinitely scalable, 99.95% up time

Our experience, your benefit – Our experience with Amazon Web Services frees you to focus on your application development

Configuration and customization freedom – There are no restrictions to continuing your customization and solution development in the cloud

Advantages of the cloud –

- No costly hardware or complex version controls necessary
- No large capital outlay
- Quicker installations
- No under- or over-used servers
- Seamless rollout of new features and functions (optional)
- 99.95% up time
- Elastic load balancing allows your application to automatically scale up or down based upon need

	Cloud Services	On Premise Client/Server
Pricing Model	Monthly subscription	License purchase or rental
Hardware	Included	Must purchase, maintain, and upgrade
OS Licenses	Included	Must purchase
Software Licenses	Included	Must purchase
Installation	No installation	Lengthy installation process
Annual Maintenance	Included	17% of AddonSoftware cost
Security, Patches, Backup Systems	Included	Customer responsibility

AddonSoftware by BASIS International Ltd. – the **big** little software company





AddonSoftware: Ready, Set, Go Deploy!

Sound the trumpets! The AddonSoftware® Shop Floor Control module is now fully ported to the Barista® Application Framework from its legacy versions 6 and 7. This milestone marks the beginning of a host of modernizations and improvements now possible with AddonSoftware's new foundation in BBj® and Barista.

AddonSoftware by Barista offers largely the same features and functionality to the user as did the legacy versions, albeit with a much more modern look and feel and all the Barista-supplied benefits. This wasn't a pure port, however. The AddonSoftware team made several improvements under the hood to leverage Barista, the BBj SQL engine, etc. Improvements included unpacking all date fields, normalizing all database tables, and restructuring program code to eliminate line numbers, and make use of symbolic labels and structured syntax, etc.

In this article, we'll discuss the basics of the Shop Floor module and review the

other completed and available modules offered by AddonSoftware. For a quick look, see [Shop Floor Control at a Glance](#) and [AddonSoftware Modules at a Glance](#) on the next two pages.

Shop Floor Control

Shop Floor uses work orders to collect, store, track, and report the costs of material, labor, overhead, and outside purchases that relate to the manufacture of an item or batch of items. The module is fully integrated; users enter information only once and it flows into other application modules automatically, including Bill of Materials, Inventory Control, Purchase Order Processing, Sales Order Processing, and General Ledger. Shop Floor Control also maintains work order history for each manufactured part number, and each work order's history is available by transaction for as long as desired.

Barista Serves a More Intuitive Interface

The core functionality of the Shop Floor Control module is the same as the legacy version, so users of character-based AddonSoftware will feel at home. But, with a BBj and Barista foundation, the AddonSoftware user interface is much more intuitive and takes advantage of graphical capabilities not available in the past.

The Release Work Order window contains a built-in grid showing required and available quantities. Automatic highlighting of any component shortages facilitates quick and easy analysis (see **Figure 1**).

Item ID	Description	Required	On Hand	Committed	Available	On Order
1001	Chromoly Main Tube	25.0000	21.0000	1.0000	20.0000	0.0000
1002	Chromoly Chain Stay	25.0000	21.0000	1.0000	20.0000	0.0000

Figure 1. Release Work Order function incorporates a grid highlighting any component shortages



By Chris Hawkins
Software Developer

Shop Floor at a Glance

• Work Order Entry



- Shop Floor makes use of three work order categories:
 - *Inventory Work Orders* are for stocked, manufactured items. Inventory work orders explode/expand/launch automatically from bills when using the Bill of Materials module.
 - *Non-Inventory Work Orders* are for the manufacture of an item, job, or customer's order not normally stocked. Use a standard bill of material for a similar item as a starting point and modify as needed before issuing the work order.
 - *Recurring Work Orders* are for maintenance, repair, or capital project activities.
- Easily enter and track planned and quoted work orders without committing materials.
 - The Release Work Order process changes work orders from planned or quoted to open status and displays component shortages for review before committing materials. Once opened, the components required are *committed* and not available for other issues or sales. The release process also records the work order finished goods items as on order. Voilà, the inventory is always under control!
 - Schedule open work orders either forward from the scheduled start date or backwards from the scheduled completion date.
 - Additional options from within Work Order Entry allow viewing and editing (if applicable) materials, operations, and subcontracts, printing a Detail Report, Transaction History, Cost Summary, and Job Status reports, assigning Lot/Serial numbers, or making a Copy of a non-stock work order.

• Printing Work Orders



- The *Traveler* is a complete list of required materials, labor steps, subcontract requirements, and comments.
- The *Pick List* prints by Operations for more efficient staging of materials.

• Committing Material



- Materials are committed automatically when work orders are released, but the *Materials Commitment Entry* process facilitates commitment in situations where item substitutions are made, immediate production quantity differs from scheduled quantity, etc.

• Inquiry Processes



- *Dispatch Inquiry* enables viewing of work orders for a given operation, user-specified date range, and priority. It also shows planned or quoted work orders to review their potential impact on the work center schedule.
- *Load Balance Inquiry* displays a filtered, graphical view of an operation's scheduled labor compared to available hours.

• Transaction Processing



- *Create Purchase Requisitions* generates requisitions for subcontracted services. The work order indicates whether someone has requisitioned, ordered, or received those items.
- *Materials Issues Entry* issues from inventory items picked for their respective work orders.
- *Time Sheet Entry* allows the entry of labor hours accumulated against an open work order and entry of the number of production items completed at that operation; parameterized to allow entry from daily time sheets (by employee), weekly time sheets (by date), or work order travelers (by work order).
- *Cost Adjustment Entry* allows for recall and adjustment of open work order operation and subcontract transactions.
- *Work Order Close Data Entry* closes work orders; either closed *complete*, meaning that no additional units will be produced, or *partial*, meaning that a portion of the units will be completed later.

• Reporting



- *Work Order Header Report* prints all or selected work order header information. When run for all open work orders, it totals all transactions for a work in process register.
- *Shop Floor Dispatch Report* lists all work orders with outstanding production requirements by scheduling priority. Use it to identify which jobs are to be worked on for each work center.
- *The Production Exception Report* highlights what production is causing work orders to become overdue; determine if delays are due to labor, purchase orders, or to both.
- *Bottleneck Analysis Report* allows entry of a percent of utilization to report any operation scheduled over that utilization. Use it to identify which scheduled jobs are creating a bottleneck.
- *Labor Efficiency Analysis Report* reviews performance of each operation in comparison to the standards defined. Use closed work orders to get an historical perspective, open work orders to analyze the current situation, or both.
- *Cost Variance Analysis Report* analyzes the difference between standard and actual costs in operations, materials, and subcontracts. It is useful for adjusting standard labor hours, tracking the performance of shop labor and production management, and analyzing the sources of labor variances.
- *Date Analysis Report* analyzes work orders for estimated start or completion dates, actual start dates, or last active date.
- *Committed Materials Report* prints a list of material committed to a work order but not issued. The report compares committed balances for each item code to on-hand, total committed, available, and on-order quantities.

Other Modules at a Glance

Accounting



- **Accounts Payable** - Provides complete payables tracking, accounting, and management of accounts payable and check writing functions. Use a flexible payment selection for improved cash flow or vendor discounts, and project future cash requirements. *Integrates with General Ledger, Purchase Order Processing, Inventory Control, Bill of Materials, Sales Analysis, and Shop Floor Control.*



- **Accounts Receivable** - An easy-to-use system for processing invoices, cash receipts, statements, and customer reports. Accurately track and control customer history information and define information regarding inventory availability, order status, prior-shipped orders, pricing, credit information, payment history, and scheduled receipts. *Integrates with General Ledger, Sales Order Processing, Bill of Materials, Shop Floor Control, and Sales Analysis.*



- **General Ledger** - Provides automatic posting of transactions with easy data access and full-featured financial report generation. *Integrates with all other AddonSoftware modules.*

Distribution



- **Sales Order Processing** - Handles quotations and orders in a simple, straightforward manner. Answer questions quickly about order status, special pricing, inventory availability, scheduled purchase order receipts, open order tracking, sales, blanket order processing, and payment history. *Integrates with General Ledger, Sales Analysis, and Shop Floor.*



- **Purchase Order Processing** - Create both requisitions and purchase orders. Identify potentially late shipments, recall a list of purchase orders for any item or vendor, and keep informed of all outstanding items by vendor or item. Maintains complete purchase and receipt histories. *Integrates with General Ledger and Shop Floor.*



- **Inventory Control** - Online updating of inventory balances gives access to current details about availability, future demand, and expected receipts. Full, perpetual tracking of finished goods, raw materials, and components. Safety stock, EOQ, and order point are automatically calculated for each inventory item. Full lot or serial number tracking maintains complete history. Integrates with Accounts Payable, Accounts Receivable, General Ledger, Sales Order Processing, Purchase Order Processing, Sales Analysis, Bill of Materials, and Shop Floor.



- **Sales Analysis** - Gain fast, accurate sales information for up to two years with this module. Analyze sales, cost of goods sold, and gross profit for customers, products, vendors, salespeople, and territories. View monthly sales information or a complete sales history for any customer or product. Rolling 12-month sales reports for customers and products are also available. *Integrates with Accounts Payable, Accounts Receivable, Sales Order Processing, Inventory*

Manufacturing



- **Bill of Materials** - Designed for companies that assemble finished goods and manufacture most of the components. Provides unlimited levels of material usage and can easily develop material costs, direct costs, and overhead labor costs. *Integrates with Accounts Payable, Accounts Receivable, General Ledger, Sales Order Processing, and Shop Floor.*



- **Shop Floor Control** - Automatically schedules work orders and tracks them as they move through the shop floor. Monitor efficiency and utilization factors by reviewing the progress and current status of each job. Provides up-to-the-minute information about production activity. *Integrates with Accounts Payable, Accounts Receivable, Bill of Materials, General Ledger, Sales Order Processing, and Purchase Order Processing.*

Load Balance Inquiry uses a BBJChart control to show scheduled vs. available hours, as shown in **Figure 2**.

Figure 3 shows a sample of the the Shop Floor Calendar, which is now much easier to view and maintain with an editable grid control.

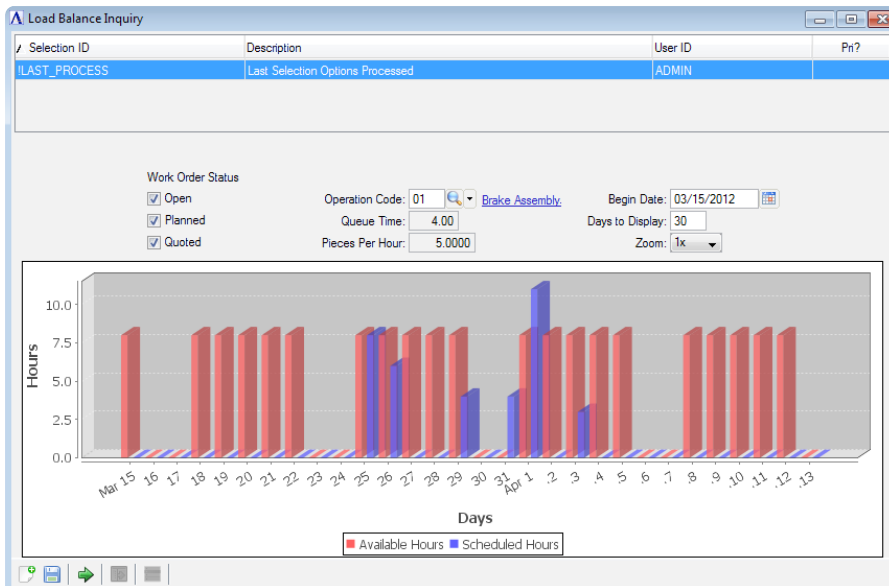


Figure 2. Load Balance Inquiry offers real-time filtering and displays results in a graphical chart control

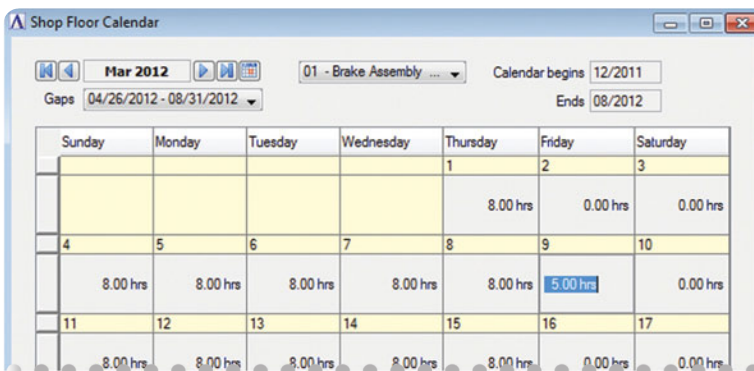


Figure 3: Shop Floor Calendar maintenance

Other AddonSoftware Modules

The AddonSoftware building blocks are bundled into Accounting, Distribution, and Manufacturing. While Payroll is not part of this suite, Payroll version 6 or 7 can run in a hybrid fashion from within Barista.

Summary

With all of the bundles and modules complete, AddonSoftware is now primed to provide the building blocks you need for your vertical-market solution. And thanks to the capabilities of Barista and Git (see the 'Git'-dy Up, Developers at links.basis.com/12git) for allowing and managing customizations – even through upgrades – you can easily, efficiently, and confidently satisfy customer requirements without the fear of being “stuck in time.”



- View a short video demonstrating AddonSoftware Shop Floor Control at youtu.be/K6NzWZs0z10
- Download AddonSoftware by Barista at links.basis.com/getaddon

Make Your Web App Sizzle With CSS

Attend this all new
training class

May 17th
after TechCon2013

Learn how to
go from this...



...to this!

Register today
links.basis.com/tcreg

The Magical Reusable Dialog Wizard



The Merriam Webster dictionary defines dialogue or dialog as *"a conversation between two or more persons; also: a similar exchange between a person and something else (as a computer)."* It is in this second context that this article will delve further and describe a new BASIS tool that allows for the creation of the program that manages a dialog between a user and a computer.

The new **Dialog Wizard**, accessible from the IDE, is a modern and powerful alternative to AppBuilder. Its key advantage over AppBuilder is that it generates fully object-oriented BBJ programs from a resource file, automatically generating the callbacks and the method stubs. But, quite interestingly, it also preserves custom editing of the resultant BBJ code, allowing you to run a second or third pass of the code generator to leverage any changes you may have made to your resource file. It also optionally generates the automatic tool button integration to Barista. In addition, the Dialog Wizard gives you the ability to automatically test the resultant code, offering a launcher to launch a GUI or BUI version of your code. And, if that wasn't enough, it also generates Javadoc-like documentation from comment lines in the code!

Let's go through the steps and see just how quick it is to go from a resource file to a running program.

Resource File to a Running Dialog Program

After creating the resource file for a simple customer maintenance form, right-click on the resource name in the Filesystems navigation tree of the BASIS IDE and choose "Run Dialog Wizard" as shown in **Figure 1**.



By Ralph Lance
Software Engineer

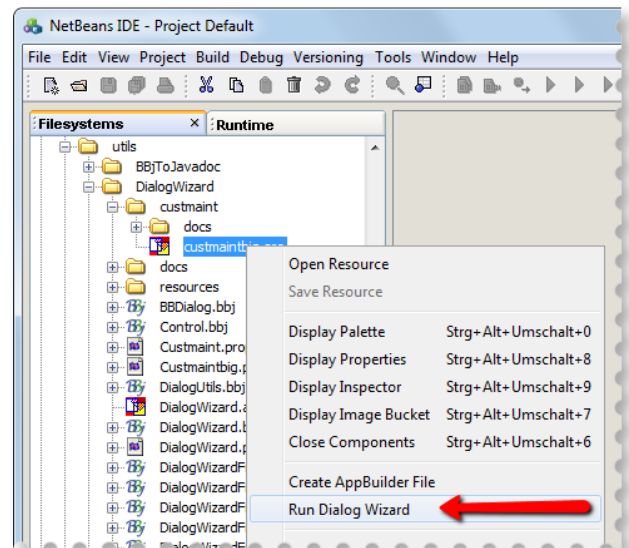


Figure 1. Filesystems option to Run Dialog Wizard

The Dialog Wizard launches and offers a suggestion of the program path to the program that is to be generated as shown in **Figure 2**; choose among the top level window IDs to steer this session of the wizard and then click [Next].

The Dialog Wizard's second frame shown in **Figure 3** lists the control names derived from the controls in the resource file and allows selection of the access scope of the member field, the variable name (or class member field) to be used in the program, the basic type of the variable, and a flag for whether the field is a required input. Click [Next].

This utility, as a minimum, requires a definition for the window close event, so select the control name that represents the top level window and either double-click on the "Close" event, or select the "Close" event and click on the right arrow button to add the Close event to the list of program events, as shown in **Figure 4**. Click [Next].

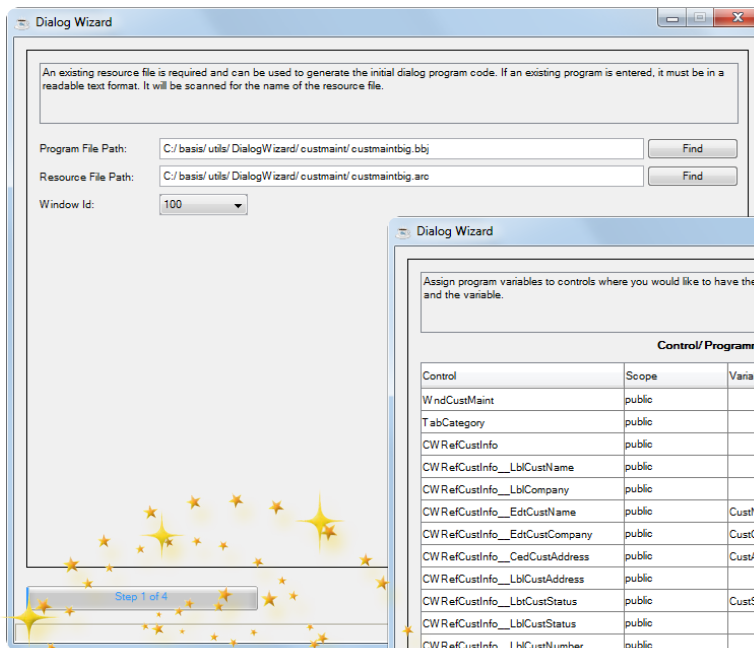


Figure 2. Dialog Wizard file paths

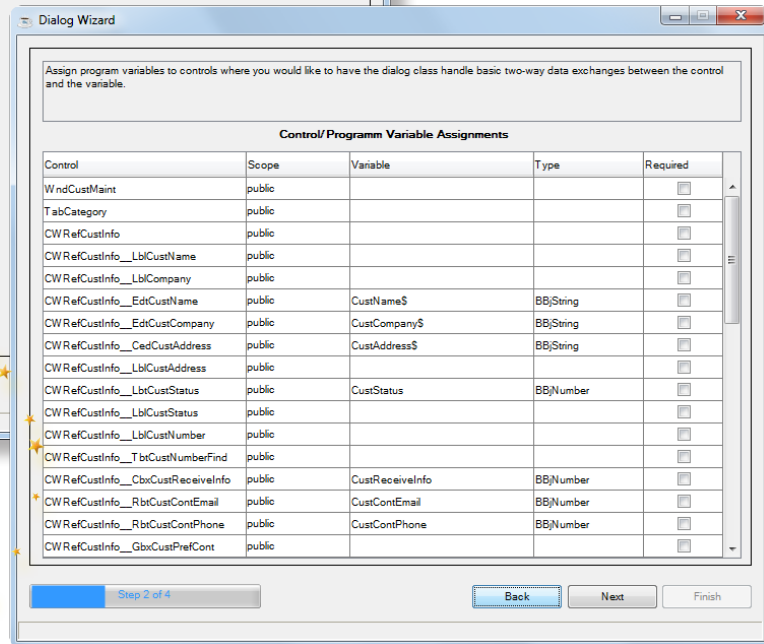


Figure 3. Control names from the controls in the resource file

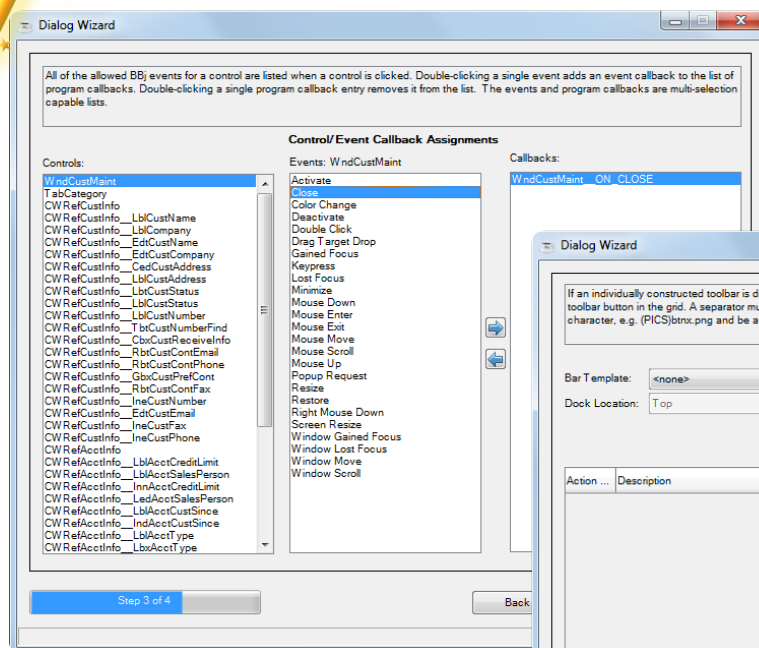


Figure 4. Add the Close event

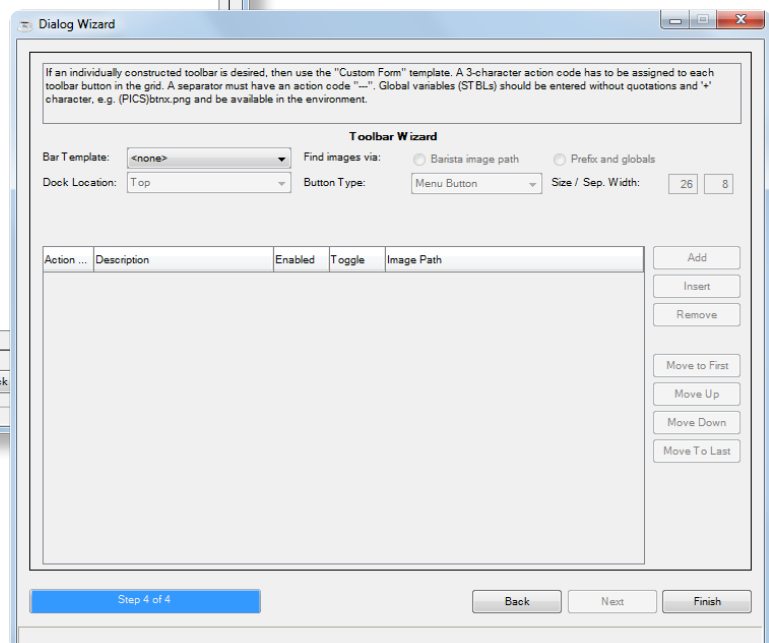


Figure 5. The final panel of the Dialog Wizard

The last panel (Figure 5) offers some selections that we'll visit a little later. For now, skip those by clicking [Finish] to generate the BBj custom dialog class program.

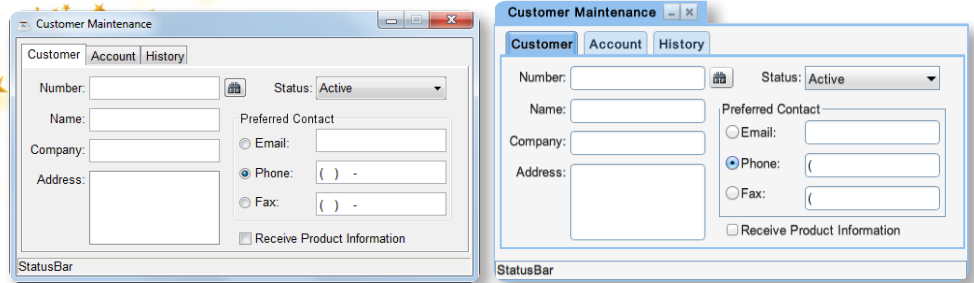


Figure 7. Test samples in GUI (left) and/or BUI (right)

Use a Dialog Wizard-generated Class in Conjunction With Barista

The Dialog Wizard is a productivity tool for individual dialogs that optionally can work with Barista. In other words, Barista isn't required to use the the Dialog Wizard or to run dialogs generated by it but it allows for the seamless integration of the dialog programs into Barista when so desired.

To illustrate this process, we'll run the Dialog Wizard again and make some changes in the toolbar section. We'll also add some code to an "Options" toolbutton to mimic an application enabling or disabling various toolbuttons to fit the context of the application.

First, we select where we would like to place the toolbar. In our example, we'll place it at the top of a maintenance form (see arrow #1 in Figure 8). In order to show the interplay between the toolbar in our dialog and the Barista MDI, we change the type of button used in the toolbar from menu to toolbuttons (arrow #2 in Figure 8). With this change, we can take advantage of the toggle function (arrow #3 in Figure 8) for showing interaction between our dialog class and Barista.

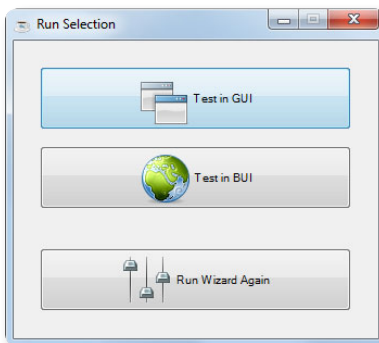


Figure 6. Test choices

One thing to note is the Dialog Wizard does not use data files, per se, for settings entered or chosen in the wizard for program generation. The program code itself contains this data. Through the use of identifiable begin and end code-blocks, for example, `rem /** DLGWIZ_DDX_BEGIN **/` and its matching end-block `rem /** DLGWIZ_DDX_END **/`, the wizard knows where to look for this data and parses what it needs from the program code. Any code outside of these blocks is ignored and will remain intact during the regeneration process.

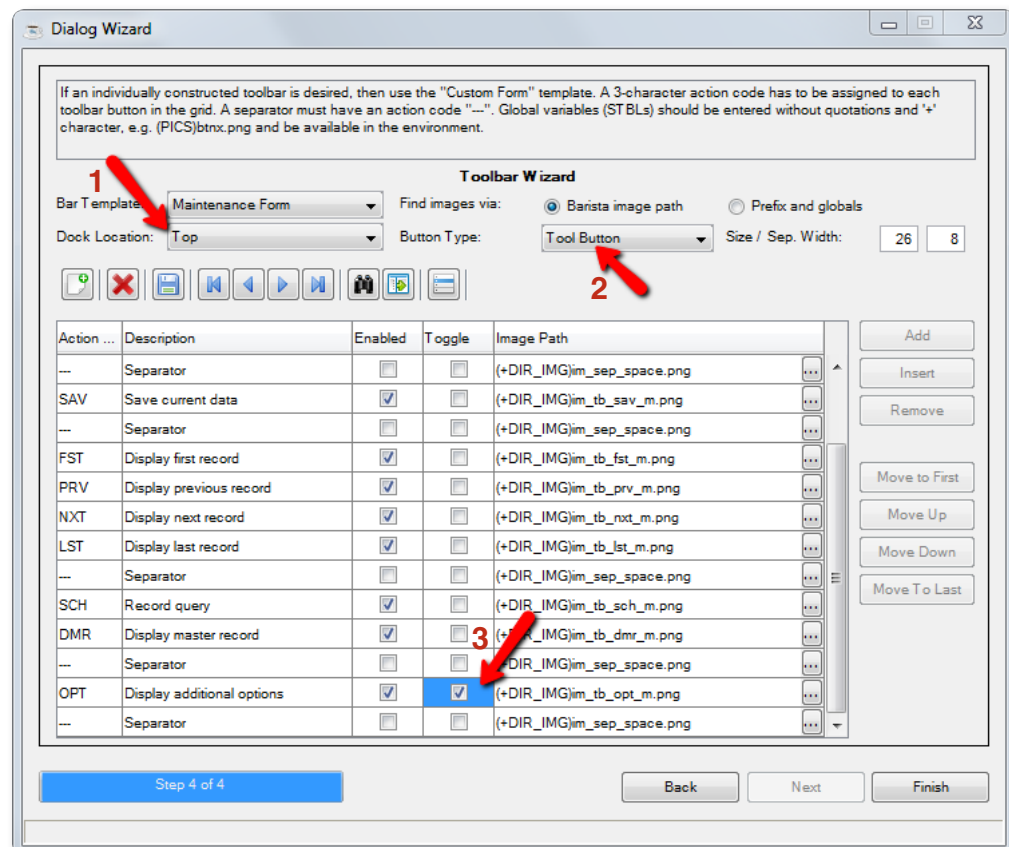


Figure 8. Defining the toolbar for Barista integration


```

rem /**
rem * Method onTBarBtnLSTPush:
rem * Event handler for the LST toolbar button
rem * @param BBjToolButtonPushEvent Tool Button Push
rem */
method public void onTBarBtnLSTPush(BBjToolButtonPushEvent pEvent!)
methodend

rem /**
rem * Method onTBarBtnSCHPush:
rem * Event handler for the SCH toolbar button
rem * @param BBjToolButtonPushEvent Tool Button Push
rem */
method public void onTBarBtnSCHPush(BBjToolButtonPushEvent pEvent!)
methodend

rem /**
rem * Method onTBarBtnDMRPush:
rem * Event handler for the DMR toolbar button
rem * @param BBjToolButtonPushEvent Tool Button Push
rem */
method public void onTBarBtnDMRPush(BBjToolButtonPushEvent pEvent!)
methodend

rem /**
rem * Method onTBarBtnOPTPush:
rem * Event handler for the OPT toolbar button
rem * @param BBjToolButtonPushEvent Tool Button Push
rem */
method public void onTBarBtnOPTPush(BBjToolButtonPushEvent pEvent!)
methodend

classend

```

Figure 9. Generated code for the toolbuttons


```

rem /**
rem * Method onTBarBtnOPTPush:
rem * Event handler for the OPT toolbar button
rem * @param BBjToolButtonPushEvent Tool Button Push
rem */
method public void onTBarBtnOPTPush(BBjToolButtonPushEvent pEvent!)
    actionList$="FSTPRVNXTLST"
    if pEvent!.isSelected() then
        #disableToolbarButtons(actionList$)
    else
        #enableToolbarButtons(actionList$)
    endif
methodend

```

Figure 10. Code to enable/disable the toolbar buttons

Click [Finish] to regenerate the program automatically. The code shown in **Figure 9** now contains method stubs for all of the toolbuttons on the toolbar.

Next, we add a little bit of code shown in **Figure 10** to the "Options"  toolbutton to enable and disable the four navigation buttons when the "Options" button is clicked.

These two methods are available for enabling and disabling toolbar buttons in the dialog as well as the corresponding toolbar button and menu item, for running the dialog class in Barista.

The new code lives outside any Dialog Wizard begin and end blocks, and is therefore preserved between program generations. If we now run our program in GUI, and click on the [Options] button, we will see the record navigation buttons toggle between an enabled and disabled state. Additionally, if we run this class inside of Barista (see **Figure 11**), we also see that the corresponding toolbuttons and menu

items in the Barista MDI match the enabled state of the navigation buttons on our form.

Summary

The new Dialog Wizard is great for rapid application development of your GUI apps, your BUI apps, and your custom dialogs that may not fit well into the core Barista data dictionary driven paradigm. You can create hand-crafted code and very easily plumb it into Barista. The Dialog Wizard encourages good development practices, as it results in documented object oriented code. It is also flexible enough to handle multiple executions against resource files that change over time as the program matures. See for yourself how helpful it can be on your next coding project! ■



See Dialog Wizard in the online documentation at links.basis.com/dialogwizard

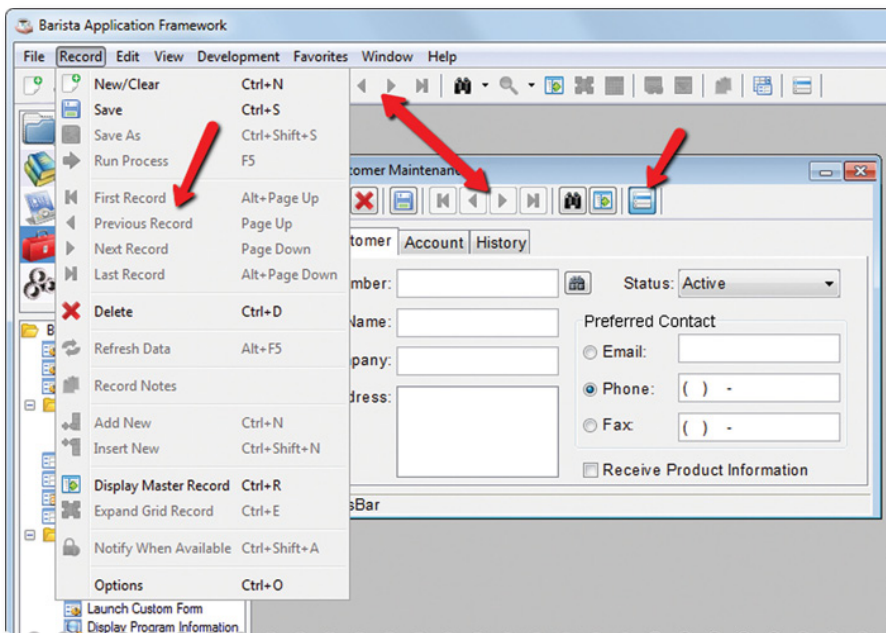


Figure 11. All record navigation buttons are disabled as a result of pressing [Options]



Going *Fast, Faster, Fastest*

BBBj® 12 includes significant optimizations to the string handling code and to various verbs and functions. Java 7 also includes some enhancements to the operation of the Just-In-Time (JIT) Compiler. Often, a given optimization will only apply to a specific portion of the code, or a particular programming idiom, but the optimizations included in the release of BBj 12 and Java 7 are unique in that they apply to virtually every program that runs on the BBj platform and on the Java platform.

Consider that in BBx®, string values and string variables are used on nearly every line of code. If we improve string performance even just a small amount, it will improve the performance of a large number of lines of code in a BBx program. Similarly, if we improve the performance of a large number of functions in the language, every line of code that uses a function has the potential to see some improvement. Optimizations to these core language features enhance the performance of every program written in BBx.

The release of Java 7 brings similar improvements. The Java Virtual Machine (JVM), on which the BBj platform runs, has a runtime component called a JIT Compiler. The purpose of the JIT compiler is to transform code from a platform-independent interpreted format into instructions specifically tailored to the underlying hardware. Improvements to the JIT compiler affect all Java-based programs, including BBj. This article will explain some of the optimizations and show some of the results of the efforts BASIS took to make your code run faster.

Java 7 Improvements

Research into the operation of typical Java programs has shown that the vast majority of time executing code is spent in a small fraction of the entire codebase, and BBj is no exception. The code that the Java Virtual Machine exercises is partly dependent on BBx code interpreted by the BBj interpreter, but BBj performs common operations on every line of BBx code. As the JIT compiler compiles those operations into native code for the host platform, those operations BBj executes most frequently benefit from highly optimized code specifically targeted for the host OS and hardware.

Since BBj Services usually runs for a long period of time, often running many thousands of interpreter instances, the JIT compiler has time to continue to optimize even those portions of code that are not executed as frequently. Over time, the JVM adapts to the characteristics of your programs and tailors its optimized, generated native code to run your program as fast as possible, identifying the most likely code paths taken by the BBj codebase when running your code and ensuring those paths run as quickly as possible.

In the late versions of Oracle's release of Java 6 and with improved performance in Java 7, the JIT compiler added an important optimization called "escape analysis."

Escape analysis allows the JVM to eliminate unnecessary synchronization and allocation. If the JVM can determine a hard upper bound on the lifetime of a particular object, it has the opportunity to avoid allocating the object at all. With fewer constraints, it can also guarantee the object is never shared between two distinct threads of execution. That guarantee allows the JVM to eliminate any locking done on an object. All programs benefit from avoiding unnecessary work, BBj included, and since BBj often allocates objects that are never shared between two threads, eliminating unnecessary locking can produce visible gains in your BBx application.

BBj 12 Improvements

Of course, these improvements only indirectly affect BBx code by improving the performance of the BASIS codebase. At the same time as the engineers at Oracle have been improving the performance of Java code, BASIS engineers have been busy at work directly improving the performance of BBx code. Here is some insight into what goes on behind the scenes as we improve our codebase, and ultimately, the performance of your application code.

Soliciting Samples

In the summer of 2011, we sent a request to the developer community for samples of code that showed opportunities for performance improvements. In addition, we identified a few specific areas through our customer support transactions that we wanted to address with these changes in the string handling code. We handcrafted several tests that executed very specific individual operations ranging from common functions and verbs to various operators. We also wrote a program generator that could automatically generate programs that select from a wide variety of normal string-related



By Adam Hawthorne
Software Engineer

operations, including familiar ones like substring and concatenation. All these samples allowed us to have baselines from a wide array of sources so we could compare with previous versions of BBJ to ensure we did not introduce any significant performance regressions with our improvements. However, in order to perform a proper comparison between the old code and the new code, we needed a test harness that could paint an accurate picture of the performance of these samples.

Performance Testing Framework

When getting a sense for how fast something is, it is often tempting to write a quick benchmark, run it a few times, and calculate a simple average using the arithmetic mean. Repeating this a number of times, the inaccuracy and noise evident in that kind of data becomes readily apparent (especially when trying to reproduce results to show-off the performance improvements to the management team!). BASIS' continued reliance on the cloud for its vast computational resources makes it easy to run several tests, but attempting to produce reproducible results in an environment subject to such a high degree of external influence proved a challenging task in and of itself.

An Insider Look at BASIS Testing at links.basis.com/12testing discusses a client/server based testing tool developed specifically for running BBJ programs simply known as "the testbed." After making a few improvements to this internal tool, we adapted our sample programs to run in that environment and to execute several hundred times each, for each run of the performance test suite. We use a statistical method called bootstrapping to identify a range of median values and produce a reliable confidence interval for the median running time of each test, which allows us to determine to a reasonable degree of certainty both that our timing values are consistent and whether our results show a statistically significant increase (or decrease) in performance.

Finally, certain external effects such as the JIT compiler and the garbage collector tend to skew later numbers in the tests, so we modified the performance test suite to perform a dry run through all the tests to allow those effects to settle before finally recording the results of our tests.

Making it Faster

Having a way to measure the performance is certainly an important part of our optimization efforts, but actually improving the performance of the code requires its own effort. Part of that is finding where we can improve the code.

Let's look at an example that illustrates one of the motivating issues we found. To avoid the overhead of copying bytes with every assignment, BBJ string values will share a buffer with other strings when possible. This sharing occurs in several ways, but by instrumenting our internal buffer handling code, while running some of our test cases, we discovered some opportunities to enhance our buffer sharing implementation. In the following figures, each group of cells represents a buffer of bytes that multiple BBJ string values refer to. The shaded portion of each cell represents the portion of the buffer used by the corresponding shaded expression in the associated program text.

Originally, a typical program might have code that looks like this:

```
1: a$ = "xxxxxxxx"
2: b$ = a$
3: c$ = b$(6,3) + "xx"
```

After line 3, the buffer's contents appear as in **Figure 1**.

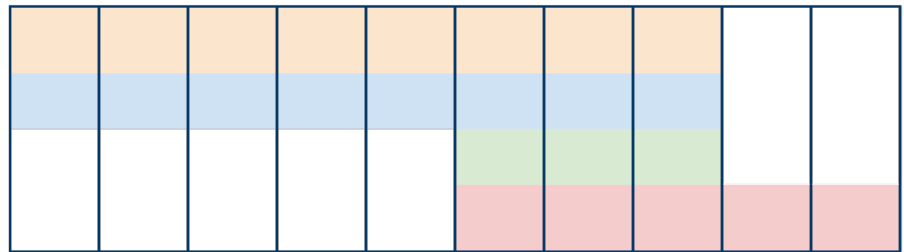


Figure 1. Buffer contents after line 3

```
4: b$ = c$
```

After line 4, our buffer sharing code would ensure that a\$, b\$ and c\$ were using the same buffer at the end of the code which avoids having to copy the buffer (**Figure 2**).

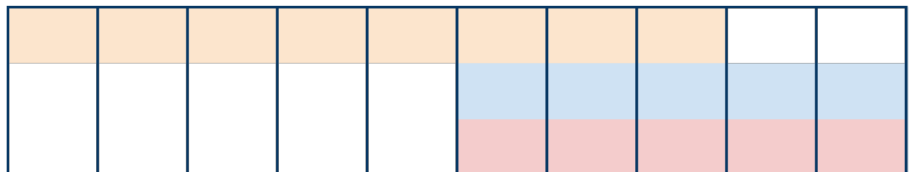


Figure 2. Buffer contents after line 4 (before optimizations)

The optimization opportunity presents itself when there is not enough room in the buffer in line 3 above to append the value "xx". In this case, in order to continue to be able to share the buffer, the original buffer handling code would copy the entire contents of the buffer above to a new larger buffer, copy in the contents of the appended string, and modify a\$, b\$, and c\$ to refer to the new byte buffer. Since some copying must take place to allocate the new buffer, it is not necessary for the new buffer to continue to share the contents with the old buffer. After making this change, the buffers look like **Figure 3**.

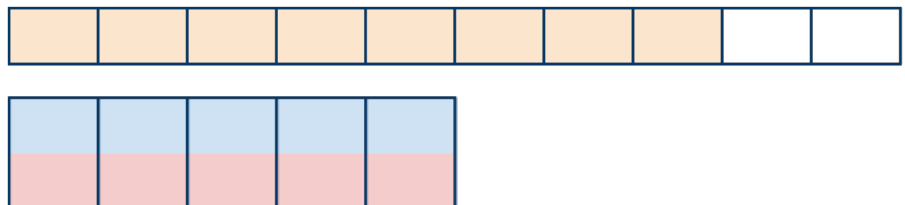


Figure 3. Buffer contents after line 4 (after optimizations)

Another benefit arose because of this change. When there are no longer any references to a\$, we no longer need to maintain the memory allocated for that buffer and we can eliminate it. This initially requires a little more memory to maintain two buffers, but eventually, it pays off because of the ability to reclaim older buffers.

In a certain code pattern, this optimization was extremely effective because we performed this kind of operation in a loop. In this case, the beginning of the buffer (the portion referred to by `a$`) would continue to grow. The buffer management code cannot track the extent of each use of a buffer without suffering an extreme performance penalty, and so the act of copying the buffer in order to append would copy hundreds or even thousands of bytes that were no longer in use.

Papercut Optimizations

We also discovered other optimization opportunities. Many of these optimizations are only noticeable on a very small scale, but we found and implemented over a thousand of these optimizations throughout the codebase. We termed them "papercut" optimizations because individually they are not very serious, but when considered all together, they can produce a significant improvement in a program's execution time. We were able to eliminate unnecessary temporary buffers in several hundred locations throughout the BBj codebase. We tightened up certain BBx function implementations to eliminate unnecessary allocation and we improved the algorithms behind some of the common BBx functions and verbs.

Results

The following charts are some results from our performance testing. Each bar in the graph is a specific performance benchmark as described previously.

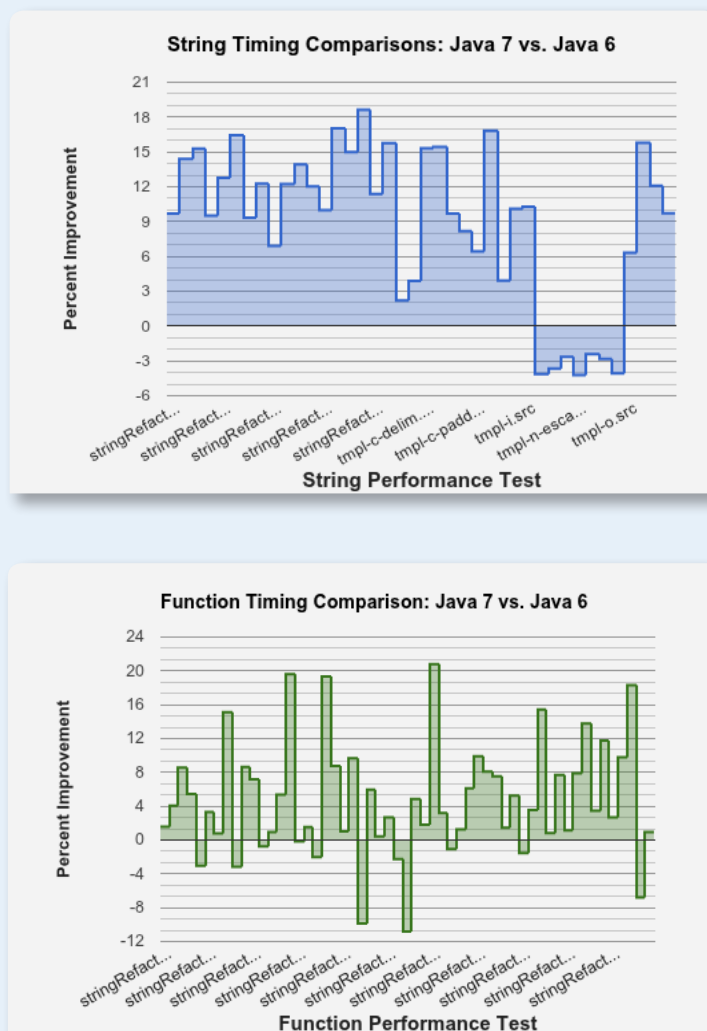


Figure 4. Performance comparisons in Java 7 vs Java 6

Figure 4 shows the differences between BBJ 12 on Java 7 and BBJ 12 on Java 6. For the vast majority of our tests, the JIT compiler improvements in Java 7 significantly enhance performance even after our optimizations. There are a few performance regressions in Java 7, but it is important to note that these are micro-benchmarks, and each individual test is unlikely to show a significant difference in a full application. These charts together show that the overall improvements are very likely to outweigh any individual regressions.





Turn On Data Auditing

Most companies have a number of employees running their applications on a daily basis. While the application or database may manage who has access to the system, legislation such as HIPPA or [Sarbanes Oxley](#) may mandate monitoring write or remove operations made to one or more data files accessed by the application. Change Audit Logging in BBJ® makes the process of building an audit trail fast and easy to implement.


How It Works

Change Audit Jobs consist of a job name, location for the audit log database(s), a list of one or more directories and/or data files to be monitored, and the frequency at which the log database should rollover. When an application changes a monitored data file using direct file access calls from a BBJ program or via SQL, the auditing system logs the change and type of change to an "audit log database." At any time, an administrator can query the log database using the interface built into the Enterprise Manager or query the log database tables directly using SQL.

When using change auditing, overhead is typically minimal since in most cases administrators configure jobs to use

the default asynchronous mode. In asynchronous mode, the audit system adds audit details to a background queue rather than waiting for the log operation to complete before continuing. In synchronous mode (not recommended), it completes the logging of the audit details before allowing the original write or remove operation on the file to complete.

Creating a Change Audit Job

To create a change audit job, select the "Auditing" item in the Enterprise Manager (EM) navigator to display the list of currently configured audit jobs. Click the  button to create a new job. **Figure 1** shows the "Audit Job Configuration" dialog.

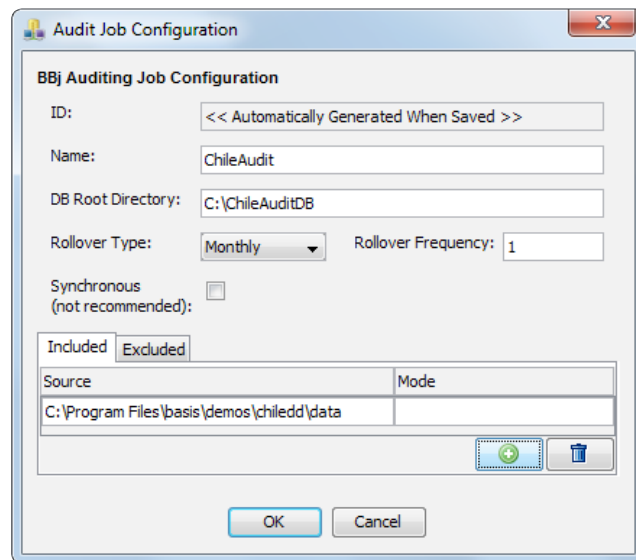


Figure 1. Audit Job Configuration dialog

Figure 2 shows the audit job from **Figure 1**, in the list of available audit jobs. "Type" indicates it is an audit job; "Last In Sync" shows the time the audit job was last in sync with all audit changes. "Running" displays YES or NO to indicate whether the job is actively running. If the job is paused, audit actions still accumulate in a queue but are not written to the audit database until resuming the job.

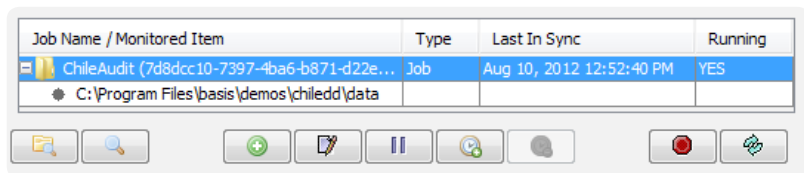


Figure 2. Audit Job List panel



By Jeff Ash
Software Engineer

Since the auditing system stores audit log messages in a BBJ ESQL database, the database list in the EM shows the auditing databases with all the other databases.

Audit job databases use the name of the job followed by the date created and a counter if there are more than one for a given date.

Figure 3 shows an audit database in the list of databases.

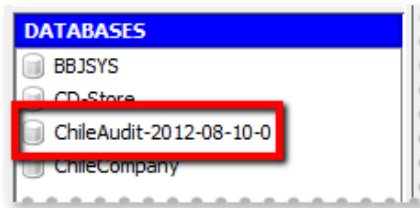



Figure 3. Auditing Database

Viewing the Audit Log Data

Audit logging data resides in a BBJ ESQL database so there are two ways to access the data: the Audit Log Viewer in the EM, or querying the database directly using SQL. We won't go into the SQL option in detail except to say that the auditing system logs each type of operation to a different table in the audit database. The Audit Log Viewer shown in **Figure 4** makes it easy to search and access this data. Use the  button to query the audit database.

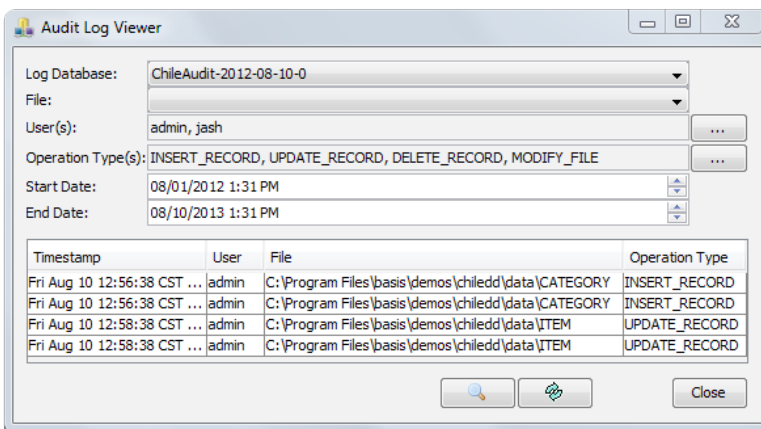


Figure 4. Audit Job Viewer

In addition to general information about each audited operation, the viewer provides drill-down support to further investigate the details of each change. Double-clicking on an operation in the viewer opens another dialog which displays the record details. For example, an UPDATE_RECORD operation shows the old record and the new record after the change, while an INSERT_RECORD operation shows only the new record added to the file. A string template entry box makes it easier to evaluate the record data since the audit operation stores the record data in its raw format. **Figure 5** shows what the user sees when viewing an UPDATE_RECORD detail.

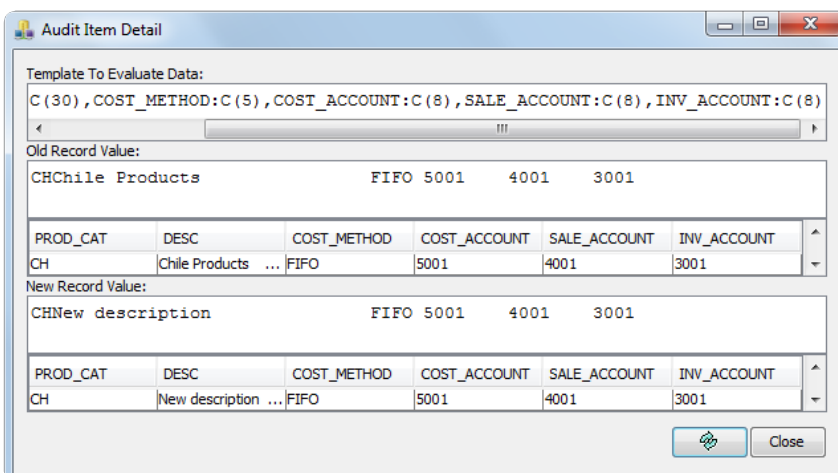


Figure 5. Audit Operation Detail

Additional Benefits

There are additional benefits of using a BBJ ESQL database for the log. First, it gives the administrator the ability to configure user access permissions to the audit database in the same way one would configure user permissions on any other BBJ database. The other notable benefit is that using iReport or BBjasper, administrators can create more robust, limited, and/or customized reporting for others to view in an external application without the need to grant them access to the EM.

Summary

If legislation such as HIPPA or Sarbanes Oxley mandate that a company needs to monitor write and remove operations made to one or more data files that their application uses, the Audit Logging feature built into BBJ is a great option. Setup and configuration takes only minutes and does not require any new coding or even shutting down BBJ Services. Further, with the use of an SQL logging database, administrators can use the built in Audit Log Viewer, or query the data directly using SQL for even more power and flexibility. ■



For more information, read *Jazz up Your Applications – Seamlessly Embed JasperReports* at links.basis.com/09jasperreports





Barista Caffeinates a CUI App With GUI Sprinkles

Legacy applications can easily contain thousands of programs and data files created over many years. When making the decision to bring them all into the graphical world of today's applications, the task can be overwhelming if not taken one step at a time. This article gives an overview of how to begin integrating fully-functional graphical components with the least amount of effort using the Barista® Application Framework and instantly begin benefiting from the myriad of built-in features that Barista brings to your application.

To begin this overview, start with the ubiquitous grid inquiry that benefits most from a graphical user interface. For another example of integrating Barista functionality into a legacy application, read *DocOut Easily Modernizes BBx Report Output* at links.basis.com/12docout.

The "CD-Jazz Store," a simple BBx® character-based application

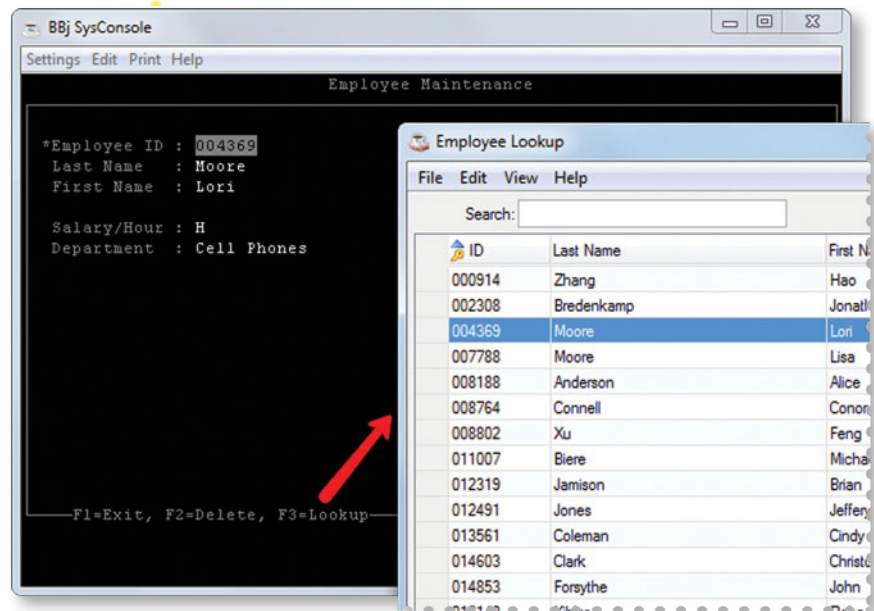


Figure 1. The CD-Jazz Store CUI program with the GUI lookup

containing 5 data files and 4 programs, is the application we will use for this illustration. The programs work together to provide minimal file maintenance capabilities for the employee data. After applying our Barista sprinkles to the CUI application, **Figure 1** shows the resulting GUI lookup launching from a keypress.

Although the user can easily navigate to the previous and next record in the CUI version using the [Page Up] and [Page Down] keys respectively, there is no provision for quickly finding one of the over 800 employees in the file if you don't know the employee's ID. This is exactly the deficit this article addresses.

Caffeinate

Accompany me as I take you through the general steps to the solution. If you would like more detail to these steps, check out the links at the end of this article and consider attending a Barista training class.



By Ralph Lance
Software Engineer

Step 1: Create a Barista application

The first step in working with Barista is to create an application. For detailed “how to” steps, refer to [Create Vertical and Customize Applications - Part 1](#) listed at links.basis.com/baristaref.

Step 2: Import the application's data dictionary into Barista

Using the “Import to Barista Dictionary” function, import the BASIS data dictionary that describes the tables of our CD-Jazz Store application. If your application does not have a data dictionary, simply make one for the table(s) you will be importing – a process that can be as simple as cutting and pasting the string template for your table (file) from your program code into the BASIS Data Dictionary editor in Enterprise Manager. The import creates element types and table definitions that Barista uses for its maintenance forms, reporting, and inquiry systems. It also creates a beginning menu for our application. For our purposes, we will just import the one employee table used in the query.

Step 3: Create a custom query

Now that we have defined our application tables in Barista, we can create the query (**Figure 2**) that we want to bind into our existing character application by running “Query Definitions” in the “Barista Development” menu and describing the columns (fields) for our query.

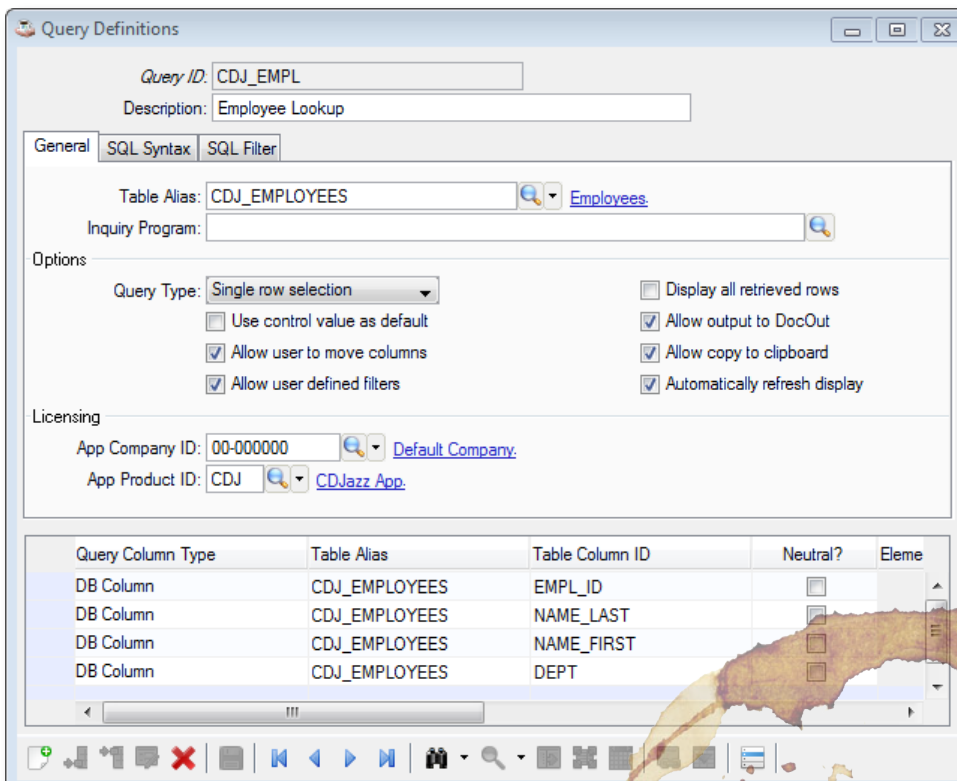


Figure 2. Create an employee lookup query

After Barista generates the SQL statement, we can immediately test the query as shown in **Figure 3**.

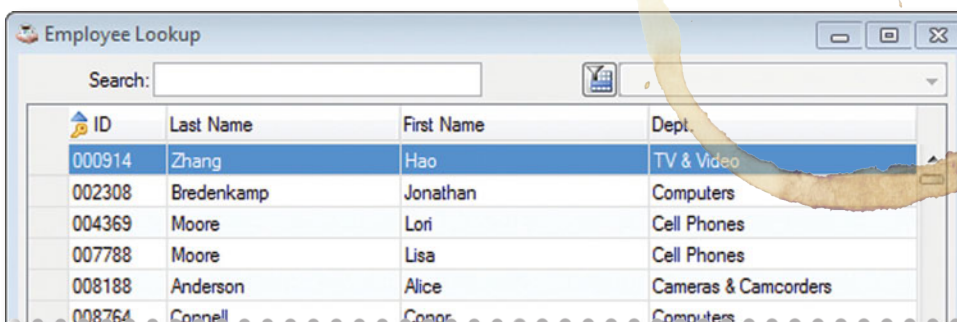


Figure 3. The result of the new employee query

Step 4: Incorporate the new lookup function into the character app

Next, we add a new [F3] lookup function to the employee maintenance screen and have it invoke our custom query via an SCALL to a Barista program and respond with the employee that the user selects. In order to retrieve the user selection from the query, we use a mechanism called a “namespace” with which two or more BBJ programs can share data and event notification.

Figure 4 shows the code block that sets up and launches the Barista query after the user presses [F3].

```
8000 REM Run Barista query
8010 bbjHome$ = System.getProperty("basis.BBjHome")
8020 pid$=hta(info(3,0)); ns_name$="query_result"+pid$
8030 a=scall(" bbj -tTO -q -WD""+bbjHome$+"barista/"" -c""+bbjHome$+
: "barista/sys/config/enu/barista.cfg"" sys/prog/bax_launch_task.bbj "+
: "- -yQ -uguest -p -qCDJ_EMPL -n"+ns_name$+" -w")
8040 selectedKey$=cast(BBjString,BBjAPI().getGroupNamespace().getValue(ns_name$,err=*next))
8050 if pos("^")=selectedKey$,-1)=len(selectedKey$) then selectedKey$=selectedKey$(1,len(selectedKey$)-1)
8060 KEY1$=selectedKey$
8070 GOTO 2260
```

Figure 4. Code to launch the Barista query

Let's take a look at this code block and dissect it by line number.

8010 determines the BBJ installation directory

8020 sets up a unique name for the group namespace variable that we would like to use for the return value

8030 invokes the Barista query via an SCALL command. The **-w** argument in the SCALL command tells Barista that we will wait until the user dismisses the query, either by closing the query window or making a selection. Read more about it in the “Barista Launch Task” article noted at the end of this article for a detailed explanation of the launch program arguments.




8040 and **8050** query the group namespace variable to retrieve the selected employee stored in its value. If the user closes the query without choosing an employee, the namespace variable will not exist (err=) or be empty. The code that scans for the caret character (^) handles multiple selections, as Barista queries optionally allow for multiple selections separated by the caret.

8060 and **8070** load the employee ID into the same input variable just as though the user had actually keyed in the ID in the CUI program.

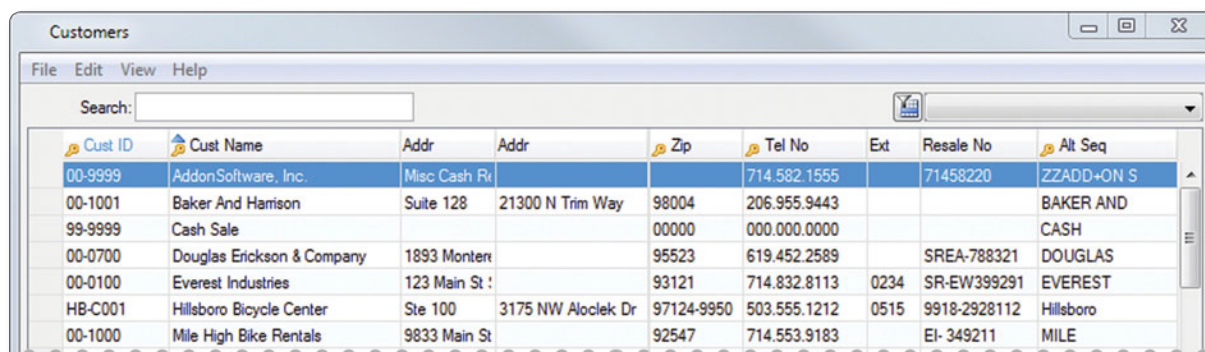
The program continues normally, displaying the employee data in the CUI program previously shown in Figure 1.

Add Sprinkles

While the ease and efficiency of this GUI lookup is worthy on its own merit, this is really just the tip of the “whipped cream”-berg. Barista provides a whole new world of query and reporting capabilities to users who are now empowered in ways not previously possible. What might have required assistance in the past from an IT person, now can easily be completed without writing programs or purchasing a third party application. In addition, this new capability offers several output types for the grid data – output to a system printer or pdf, csv, xml, xls, and txt file formats.

Barista's query for grids provide multiple methods to search, sort, filter, adjust column display, adjust column order, adjust column width, and to select all or only highlighted rows of the grid for output. The sample grid in Figure 5 shows a typical layout with the Search bar in the upper left and the filter wizard button  in the upper right. Special icons appear next to the column headers to give more information about that column; the blue triangle  shows the “sorted by” column, an asterisk would indicate that a filter has been applied to the column, and the key icon  indicates that the field is indexed, which will sort faster than non-indexed fields.





Customers

File Edit View Help

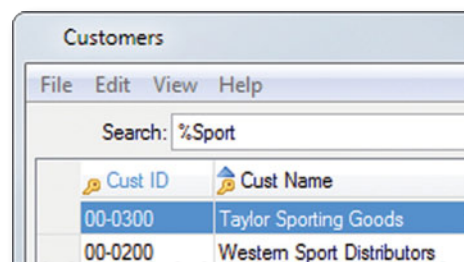
Search:

Cust ID	Cust Name	Addr	Addr	Zip	Tel No	Ext	Resale No	Alt Seq
00-9999	AddonSoftware, Inc.	Misc Cash R			714.582.1555		71458220	ZZADD+ON S
00-1001	Baker And Harrison	Suite 128	21300 N Trim Way	98004	206.955.9443			BAKER AND
99-9999	Cash Sale			00000	000.000.0000			CASH
00-0700	Douglas Erickson & Company	1893 Monter		95523	619.452.2589		SREA-788321	DOUGLAS
00-0100	Everest Industries	123 Main St		93121	714.832.8113	0234	SR-EW399291	EVEREST
HB-C001	Hillsboro Bicycle Center	Ste 100	3175 NW Alolek Dr	97124-9950	503.555.1212	0515	9918-2928112	Hillsboro
00-1000	Mile High Bike Rentals	9833 Main St		92547	714.553.9183		EI-349211	MILE

Figure 5. Sample customer master grid

Search/Sort

The Search feature creates a quick case-sensitive or case-insensitive search filter of the query grid. As soon as the user selects the column header and then begins typing the search characters in the Search bar, the filtering begins. The % is a wildcard character so entering '%Sport' will return all strings in the selected column that contain the word 'Sport' (see **Figure 6**). To restore the grid back to displaying the full result set, the user would simply clear the contents of the Search field.



Customers

File Edit View Help


Search: %Sport

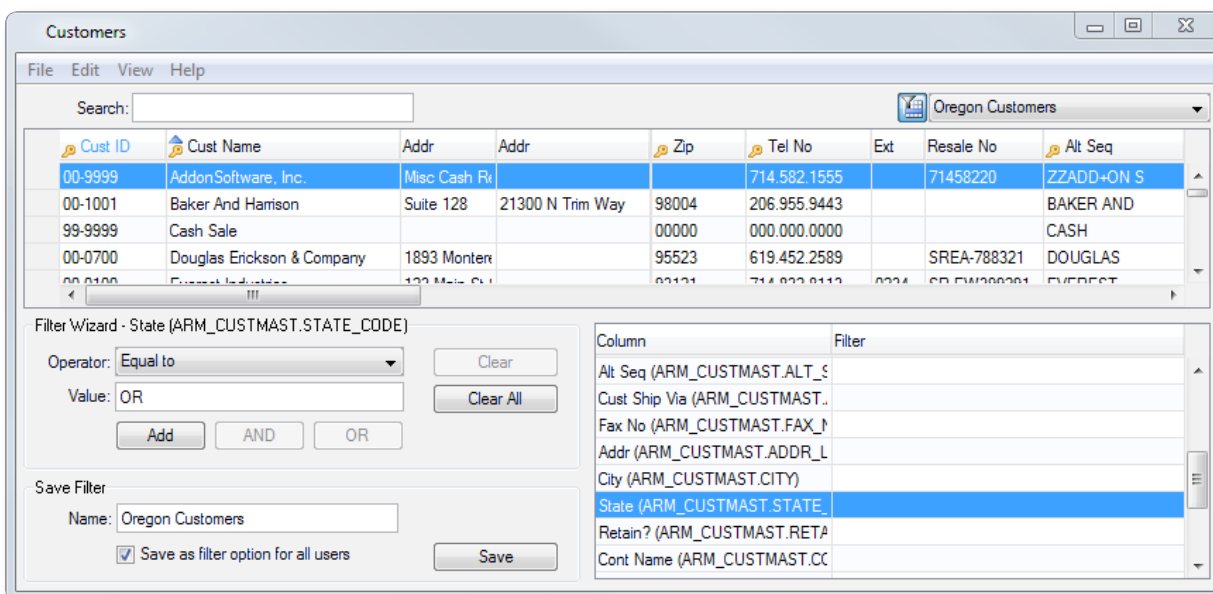
Cust ID	Cust Name
00-0300	Taylor Sporting Goods
00-0200	Western Sport Distributors

Figure 6. Wildcard search result

Users can select a single column header for an ascending or descending sort or can select multiple columns by pressing the [Shift] key while selecting the columns to sort on multiple columns.

Filter

A Filter Wizard allows users to create and save custom filters quickly and easily. Select the **[Filter Wizard]** button  in the top right corner of the query grid frame to launch the wizard. The Wizard (shown in **Figure 7**) requires user input for the Operator, Value, and Column.



Customers

File Edit View Help

Search:

Oregon Customers

Cust ID	Cust Name	Addr	Addr	Zip	Tel No	Ext	Resale No	Alt Seq
00-9999	AddonSoftware, Inc.	Misc Cash R			714.582.1555		71458220	ZZADD+ON S
00-1001	Baker And Harrison	Suite 128	21300 N Trim Way	98004	206.955.9443			BAKER AND
99-9999	Cash Sale			00000	000.000.0000			CASH
00-0700	Douglas Erickson & Company	1893 Monter		95523	619.452.2589		SREA-788321	DOUGLAS
00-0100	Everest Industries	123 Main St		93121	714.832.8113	0234	SR-EW399291	EVEREST

Filter Wizard - State (ARM_CUSTMAST.STATE_CODE)

Operator: Equal to

Value: OR

Add AND OR

Save Filter

Name: Oregon Customers

☒ Save as filter option for all users

Save

Column	Filter
Alt Seq (ARM_CUSTMAST.ALT_S	
Cust Ship Via (ARM_CUSTMAST.	
Fax No (ARM_CUSTMAST.FAX_N	
Addr (ARM_CUSTMAST.ADDR_L	
City (ARM_CUSTMAST.CITY)	
State (ARM_CUSTMAST.STATE_	
Retain? (ARM_CUSTMAST.RETA	
Cont Name (ARM_CUSTMAST.CO	

Figure 7. Filter Wizard

The Operator options appear in a dropdown list containing Greater than, Less than, Greater than/Equal to, Equal to, Not equal to, Begins with, Ends with, Contains, Does not contain, Is contained in, and Is not contained in.

To reuse a defined filter, the user may save the filter by giving it a name, and further mark whether to share the custom filter to all users in the organization. The shared filter will appear as a dropdown option next to the Filter Wizard button in the top right corner of the frame. See the filter "Oregon Customers" displayed in **Figure 7**.

Query Selection Options

Barista provides additional options when right clicking on a record in the grid (**Figure 8**). While all of these options are valuable, the Inquiry Columns function is one worth looking at more closely.

Selecting Inquiry Columns (**Figure 9**) allows users to change which columns display and in what order by marking the Show checkbox and then moving the item into the desired position using either the [Move Up]/[Move Down] buttons or dragging and dropping the highlighted column. All edits are preserved at the user level when returning to the grid. To undo selection and return to the default column view, simply click the [Restore] button.

Other Query selection options include –

- Copy - copies the columns to the clipboard, allowing the user to choose whether to include the database table/column names and text column headings, and to select the column delimiter and text identifier characters.
- Export Records - quickly exports the contents of the grid or selected columns while offering several export options; Document Output Viewer, Document Output Selection Form, as well as the type of output.

Output

After sorting and filtering and massaging the data to the user's delight, the user can easily output the results in a variety of ways by just making one or more selections in the the Document Output Selection window (**Figure 10**).

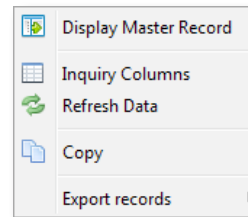


Figure 8. Query selection options

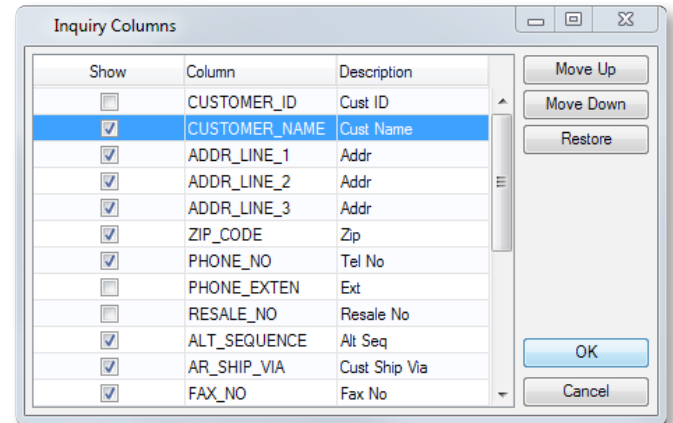


Figure 9. Inquiry Columns window

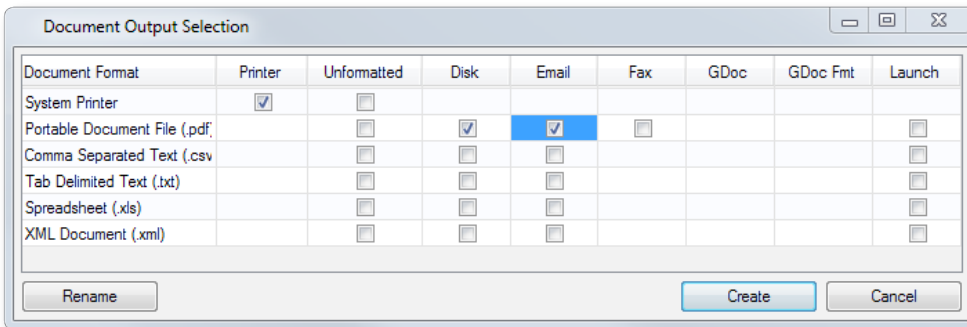


Figure 10. Document Output Selection window

Gaining all of this functionality handily meets most all of any user's simple report and output needs or wishes.

Summary

BASIS developers can use this technique to ease their users into graphical user interfaces with minimal effort and disruption to existing code by leveraging the out-of-the-box functionality Barista provides. With the addition of as little as ten lines of code to a CUI or GUI application and a few minutes spent configuring a Barista query, you can delight your users by delivering untold productivity gains to their daily tasks. In fact, with the query's sorting and filtering capabilities as well as flexible output of the grid contents or selections in various formats and mechanisms such as PDF, XLS, fax or email, users get a big bang for their buck, all with very little effort on the part of the developer. You'll be their Barista of choice to caffeinate their applications! ■



- Review *DocOut Easily Modernizes BBx Report Output* at links.basis.com/12docout
- For more information, refer to these Barista resources at links.basis.com/baristaref
 - Create Vertical and Customize Applications - [Part 1](#), [Part 2](#), [Part 3](#),
 - [Create and Synchronize Applications](#)
 - [BASIS Data Dictionary Import](#)
 - [Query Definition System](#)
 - [Barista Launch Task](#)

Compressing Apps for Zippy Network Performance

Networks grow ever more complicated. From [Ethernet](#) to [WiFi](#), consumers are connecting more devices to the Internet worldwide and in more complicated ways. Each technology differentiates itself with subtle advantages, but every networking method has limits. A network can only transmit so much information per second, which means each network has a limited bandwidth. A state-of-the-art fiber optic connection transmits an HD movie in seconds while it might take a cell phone hours to download a song over GPRS.

The high cost of networking makes efficient use of bandwidth important. Economic costs are relatively small and getting smaller, but a user paying for a cell phone data plan might disagree. For example, Amazon charges around 12 cents per transmitted gigabyte, so using those bytes wisely will save money. Cell phone providers often charge for data sent to a phone, so compression will save money. Beyond cost savings, minimizing the number of bytes transmitted saves something far more valuable - time. Users have no interest in watching screens load. The more narrow the connection, the more critical it is to be efficient. Cell phone users want their data immediately.



By Jason Foutz
Software Programmer

Your network provides access to your applications. Anyone connected to your network could launch your application with Java Web Start, if desired. If you make your application available on the Internet, people could connect with almost any device – smartphone, tablet, laptop, or desktop – to your application. Perhaps they are using a fast connection on their desktop at work. They could be relying on WiFi at a customer's job site. They might even be using a cellular connection built into a tablet from the beach. It's impossible to predict all the different types of network clients that your application must deal with.

From version 12 onward, BBJ® makes use of Java's [Pack200](#) technology, which cuts the bandwidth required for initially launching a thin client on first install or after an upgrade by over 60%. Jetty, the built-in Web Server that ships with BBJ, automatically compresses jars with Pack200 reducing, for example, BBJThin client .jar from over 6 MB to less than 3 MB. On a local network, this compression saves users a few seconds; over the Internet, it saves minutes.

BBJ's Web Server also uses [Gzip](#) compression for many resources. Gzip speeds up both BUI and static resources served from the htdocs directory. Gzip provides the biggest benefit when used on text files. All standard web files such as HTML, CSS, and Javascript are readily compressible by Gzip. Gzip does a really fantastic job with core BUI files, reducing 1.3 MB files down to less than 400 KB. The payoff of this huge saving is most valuable and most visible on smartphones. Decreasing the amount of required bandwidth saves customers significant time and money resulting in a greatly improved user experience.

Compression in BBJ happens behind the scenes. The Jetty Web Server examines your resources and compresses them as needed. Jetty compresses any jar required by the Web Start thin client before sending it over the network. Compression reduces the size of the JavaScript files that support a BUI application by 60%. Jetty also compresses additional resources such as custom CSS or a customized HTML index page. All of this happens behind the scenes without any extra effort. Jetty provides all of the speed without any administration overhead.

Summary

BASIS takes advantage of several industry standard techniques to make your business application as small and nimble as possible on any network –

- Pack200 drastically reduces network resources for downloading Web Start clients
- Gzip speeds up BUI
- Jetty compresses virtually every resource served with state-of-the-art tools to save your customers time, and money.

Compress coal and you get diamonds. Compress web resources and you get happy customers. Priceless. ■



DocOut Easily Modernizes BBx Reports

Add Print Preview, PDF, XLS, CSV, XML, Google Docs, Fax, Email, and Archiving

DocOut is the document output subsystem component of the Barista® Application Framework RAD tool with a ton of built-in benefits and features. Developers can now leverage these benefits from new or existing PRO/5®, Visual PRO/5® or BBj® code, without having to use the entire framework, and with very little programming effort!

To take advantage of DocOut, developers would simply change all of their 'PRINT @' statements to vector assignments and invoke the DocOut

object. With those changes in place, applications are upgraded to a modern reporting system chock full of new features including print preview, multiple output format options, user interactive column sizing, report archiving, and more! After converting an application to use DocOut, a pleasing print preview of the report appears similar to the one shown in **Figure 1**.

Empl. ID	Last Name	First Name	(S)alary (H)ourly	Department
000914	Zhang	Hao	S	TV & Video
002308	Bredenkamp	Jonathan	S	Computers
004369	Moore	Lori	H	Cell Phones
007788	Moore	Lisa	H	Cell Phones
008188	Anderson	Alice	S	Cameras & Camcorders
008764	Connell	Conor	H	Computers
008802	Xu	Feng	H	Computers
011007	Biere	Michael	S	TV & Video
012319	Jamison	Brian	S	Games & Toys
012491	Jones	Jeffery	H	Music, Movies & Book
013561	Coleman	Cindy	S	TV & Video
014603	Clark	Christopher	S	Games & Toys
014853	Forsythe	John	H	TV & Video
016148	Kibler	Rebecca	H	Car & GPS
016949	Fowler	Laura	H	Music, Movies & Book
017995	Jarvis	Jason	S	Cell Phones

Figure 1. DocOut report in print preview

Follow along in this review of the DocOut benefits and see the enhanced sample application.



By Ralph Lance
Software Engineer

Feature Overview

The DocOut report provides the same columnar or tabular report as the legacy program, but the DocOut system supercharges it with several extra capabilities. DocOut offers a number of output file types including PDF, XML, CSV, and XLS, shown in **Figure 2**, and puts the user in control.

After selecting the [Save] menu button, the user is able to choose the name of the saved report as well as the output type from multiple file formats. DocOut even provides an interface to Google Docs so users can save their reports in the cloud for ubiquitous access from any Internet-connected machine or device. Raw output is also an output option for those who use a third party tool to manipulate that output. Sending the report to the printer is a standard option, of course, but the DocOut system also allows the user to fax or email the report directly from the preview window.

In addition to viewing options such as scrolling through the report, sizing the output, and zooming in and out, the user is also able to affect the layout of the report directly from the preview window, as shown in **Figure 3**. DocOut gives the user the ability to adjust the widths of individual columns or even hide them if they are not applicable for the report's recipient. To aid in alignment for numeric data, DocOut can optionally modify the report to use a fixed-width font. Talk about the putting the user in control!

DocOut is flexible enough to give not only the user but also the application extensive control over the report. By adding code to automate DocOut's output settings, the app is in complete control over the report – even to the point of eliminating the user interface. This is a must-have feature for batch reports, as the application can configure and emit all of its reports without requiring any user intervention.

DocOut also offers a one-step methodology allowing the user to select and process a particular output type. Barista's Document Inquiry System is a couple of mouse clicks away, providing the ability to select multiple output types and reprint documents. Additional options are set directly from the Document Output Selection window (**Figure 4**) such as sending

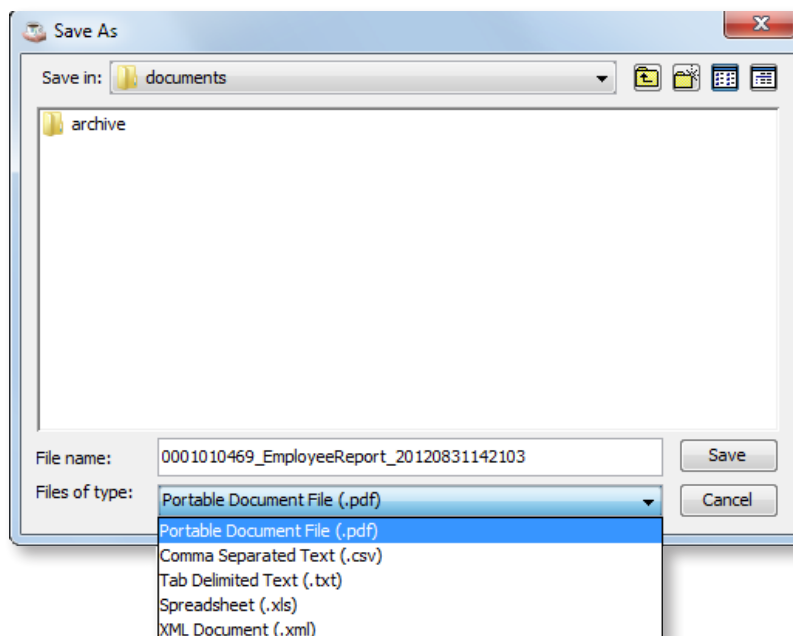


Figure 2. Save As output formats

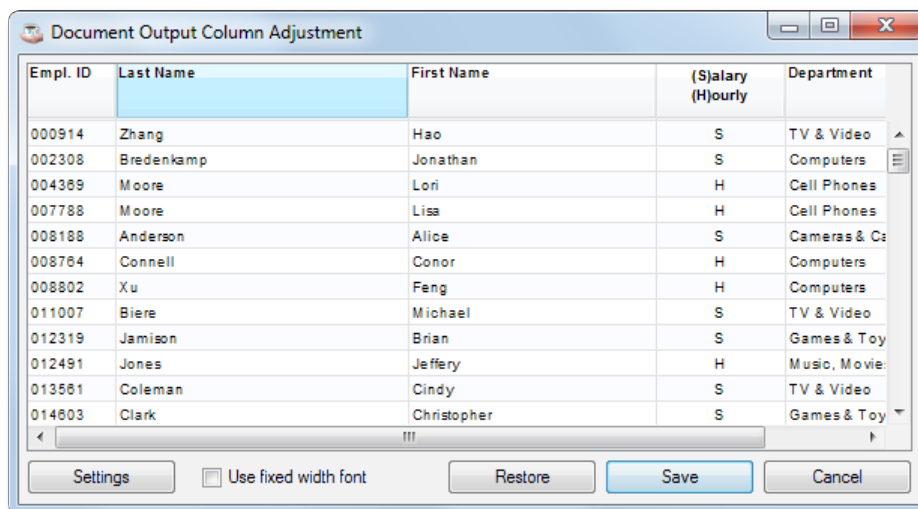


Figure 3. DocOut's print preview with output column customization

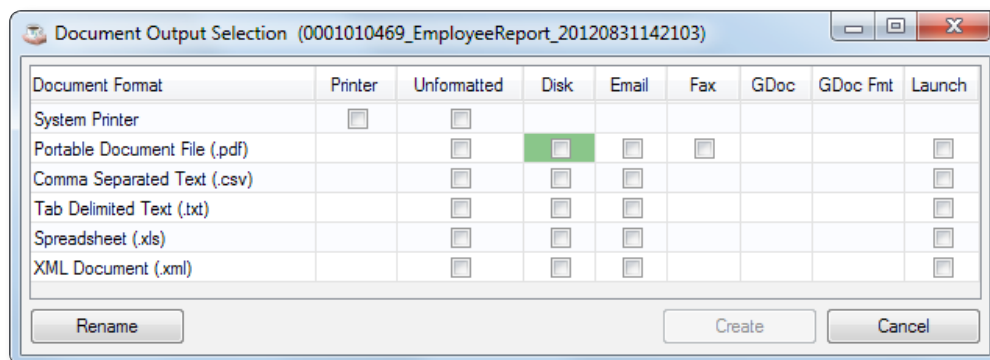


Figure 4. Document Output Selection Dialog

the document via fax or email, saving the document in Google Docs (GDoc) in the cloud, and even launching the target document once the processing has completed.

Utilizing DocOut saves significant time and money in the development effort. Kurt Williams, of Marex Services acclaims:

"It's very easy to convert an existing report to the DocOut process. Instead of using PRINT statements to send the data off to the printer, you just load the data into a BBJ Vector and let DocOut process the report. The payoff is huge. The report is rendered in a print preview window and from there, you can adjust columns, output the report to the printer, convert it to a pdf, xls or delimited text file, email or fax it to interested parties, archive it, or simply review the data in the print preview and discard it!"

Using the DocOut Object

Everything needed to create amazing reports and output them in various types, including PDF and spreadsheet files, is built into the system without the need for adding plugins or prerequisites for third party products. Developers not only maintain complete customization control, but they benefit automatically from new features added to DocOut without requiring any additional code changes to their applications.

DocOut offers multiple benefits and incorporates an abundance of functionality. But just how much coding effort is required to take advantage all of this functionality? Let's take a look. BASIS designed the DocOut Object for use by any BBJ program, obviating the need for the program to run inside the Barista framework. This approach allows legacy apps such as a CUI PRO/5 app running in BBJ, for example, to have greater flexibility when it comes to reporting.

The new code is simple and very straightforward. For a more descriptive overview and detailed analysis of the process, review the [DocOut Tutorial](#) and the [DocOut Object](#) documentation referenced at the end of this article.

Our step-by-step example uses the DocOut object to produce a report of employees for a traditional character-based application called the CD-Jazz Store. Try the examples yourself by downloading the code from links.basis.com/12docout-code.

1. Modify the program to set up the Barista environment instead of opening the printer alias.

- a. Specify the complete path to the DocOut class when outside the Barista environment.

```
use ::C:/basis/barista/sys/prog/bao_document.bbj::Document
```

- b. Otherwise, use this relative path within the Barista environment.

```
use ::sys/prog/bao_document.bbj::Document
```

2. Declare object variables used in the program.

```
declare Document doc!  
declare BBJVector out!
```

3. Create (instantiate) the DocOut Document object.

```
doc! = new Document()
```

4. Set the user authorization. Alternatively, use BBJ User Authentication setup in Enterprise Manager.

```
doc!.setUserID("ADMIN")  
doc!.setPassword("admin123")
```

5. Set the report parameters.

```
doc!.setFirmID("01")  
doc!.setLanguage("ENU")  
doc!.setDocumentID("EMPL_REPORT")  
doc!.setReportTitle("Employee Report")
```



6. Set the report headings.

```
doc!.addReportHeading("Employee Report")
```

What's the next step? Set up the report columns.

The `addColumn()` method makes this a snap. The application code passes all of the necessary information for the report columns, including the following:

- Column heading
- Base data type
- Data length
- Optional column width override in pixels
- Optional output mask to be applied
- Optional Barista control type code (for future use)
- Optional justification code (defaults "L"eft for character, "R"ight for numeric columns)
- Optional formatting flags that can be used in combination, such as setting the typeface to bold and stipulating that the value is a total and should be underlined

Again, the DocOut tutorial and documentation describe these parameters and their permissible values in more detail. To localize the headings, take advantage of BBJ's [translation tools](#).

Based on the template for the employee record example, the code to set up the report columns is pretty straightforward:

```
doc!.addColumn("Empl. ID", "C", 6, 60, "", "", "", "")
doc!.addColumn("Last Name", "C", 50, 200, "", "", "", "")
doc!.addColumn("First Name", "C", 50, 200, "", "", "", "")
doc!.addColumn("(S)alary^(H)ourly", "C", 1, 60, "", "", "", "")
doc!.addColumn("Department", "C", 20, 200, "", "", "", "")
```

The use of the caret character in "(S)alary^(H)ourly" stipulates the desire for multiple column header lines.

With the report defined, it's time to fill it. In this example, simply read through the employees file and store the individual field values as strings in a BBJVector that gets passed to the `setOutputData()` method.

If the original application existed before DocOut, then it will most likely have contained `PRINT @` statements and page handling logic. As we can see here, DocOut integration greatly simplifies this code:

```
empl_chn = unt
open(empl_chn)"d:/training/cdjazz/data/cdj_empls"
dim empl$:"empl_id:c(6*),name_last:c(50*),name_first:c(50*),sal_hour:c(1*),dept:c(20*)"

out!=BBJAPI().makeVector()
while 1
  readrecord(empl_chn, end=*break)empl$
  out!.addItem(cvs(empl.empl_id$,3))
  out!.addItem(cvs(empl.name_last$,3))
  out!.addItem(cvs(empl.name_first$,3))
  out!.addItem(cvs(empl.sal_hour$,3))
  out!.addItem(cvs(empl.dept$,3))
wend
doc!.setOutputData(out!)
```

The final step is to instruct the DocOut object to actually produce the report and release our program with this code:

```
doc!.process()
release
```

By default, the DocOut object starts the report generation process synchronously so the program will wait for its completion. To override this, invoke the `setSessionWait()` method:

```
doc!.setSessionWait(0)
```

Using the DocOut Object With PRO/5 and Visual PRO/5

With the advent of XCALL, existing Visual PRO/5 and PRO/5 applications can also take advantage of the DocOut Object to XCALL the BBJ program version of a legacy report. DocOut can either interact with the end user via a BBJ thin client session on the desktop or can be invoked in BBJ on the server-side only, with no user interface. In that case, one could still run legacy code in PRO/5 on a terminal emulator while generating the email/fax/pdf/archived document in BBJ on the server. To view the report on a PC running PRO/5 or BBJ in a terminal emulator, simply save it as a PDF in a directory on the server that the built-in Jetty Web Server can access, then provide the URL to the CUI session.

We encourage you to take a look at the documentation for the DocOut Object and the tutorial so you can retrofit your code by replacing those `PRINT @` statements with vector assignments and reap the multitude of DocOut benefits such as archiving, multiple file formats, and multiple delivery mechanisms, all with the minimum of programming effort. ■



- Download the code samples in this article at links.basis.com/12docout-code
- For more information about DocOut and the methods referenced in this article, see
 - DocOut Tutorial at links.basis.com/docout_tutorial
 - DocOut Object at links.basis.com/docoutobject
- Find BBJTranslator in the online documentation at links.basis.com/bbtranslator

New hands-on training courses!

Extend your stay in Las Vegas after TechCon2013 for training. Several of these classes are a one-time offering so don't miss the chance to put to use what you learned and saw in action during TechCon!

May 16-17

- **Barista Advanced** (Beginning Barista is a prerequisite...still time to sign up and attend the web-base class on March 18, 20, 22)

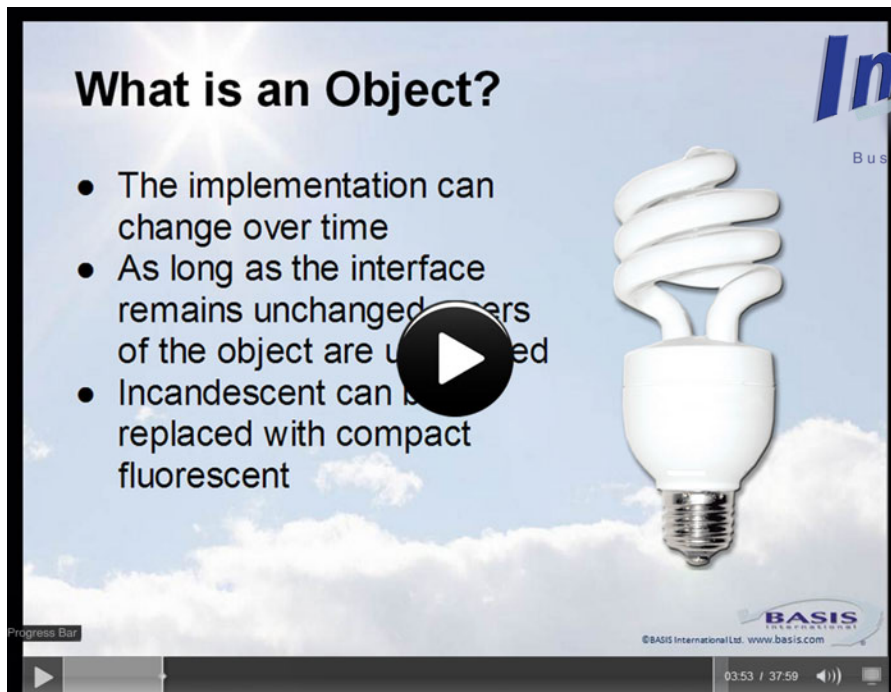
May 16

- **Eclipse IDE** – Editor, Debugger, FormBuilder, Enterprise Manager – SVN Plugin, Java Plugins, HTML (Aptana), Plugin
- **BASIS Cloud Transformer** (Ride the BASIS “lift” to the cloud) + Addon Cloud

May 17

- **BASIS CUI to GUI Best Practices** (1/2 day AM)
- **ERP Building Blocks** (1/2 day PM)
- **Make Your BYOD Web Apps Sizzle with CSS**
- **DBMS Enhancements** - No Coding Required Replication, Scheduler, Security Options (SSL), data warehouse, data recovery plan, best practices; Are you prepared? Ever restore from your back-up? Authentication options

To register, go to links.basis.com/tcreg



BASIS Unveils New Training Format

For many years, BASIS offered product training courses in the traditional onsite classroom format. Later, BASIS expanded training to reach customers in the comfort of their offices with web-based training. Realizing the effectiveness of our partially recorded Java Breaks, BASIS 'canned' some key training classes and offered this new format, blended with live monitoring and instructor interaction for questions and answers.

Moving the foundation of web-based training to pre-recorded content ensures that students receive consistent delivery of all the important points needed to understand the key concepts for each class. In a pure "live" session, trainers risk getting sidetracked and missing important concepts. Often, time gets away and the session ends before covering all the material or trainers might rush through the last hour in an effort to complete the content by day's end. Now students receive the course content completely, consistently, and at a comfortable pace.



By Amer Child
Digital Communications/
Web Developer

Much to our delight, the feedback on our first three training courses delivered in this new style was positive from students and trainers, both agreeing that this is a win-win approach. The BASIS trainers enjoyed the new format better than a live session; they could better focus on questions the students asked and even pause the video to fully address them. Students who encountered issues while working the exercises during the session still got one-on-one help from the trainers. Since the exercises were also recorded, students could pause as they needed to work through a step, or get personal assistance from the trainer. The trainer could switch from the recording to a live desktop and repeat the exercise again and, as needed, remotely access the student's desktop to "drive" through the issue. One participant of a recent course noted in his feedback, "I welcome more classes like this for other BBJ components!"

In the future, BASIS customers will see more core training courses recorded and delivered in this pre-recorded web-based format. In addition to using these recordings as the backbone to live training courses, BASIS plans to roll them out for customers to access in a self-paced "on demand" format. In the interim while we are developing this strategy, students who attend a class can access the recorded course for two weeks after the initial training class as a resource to practice what they learned.

Overall, the excitement about the new training strategy is high among customers and BASIS trainers. We look forward to your participation in future sessions and to hearing about your experience and your requests for future topics that use this new training format! ■



New BASIS Training

Feb 7.....	BASIS DBMS With SQL, Stored Procedures, Triggers, Replication
Feb 21	Object-oriented Programming With BASIS Custom Objects
Mar 7	Report Writing With iReport and BBJasper
Mar 18, 20, 22.....	Beginning Barista Application Framework (required to attend the Advanced Barista class on May 16-17)

Traditional Classroom Training

May 16-17..... Post TechCon2013 in Las Vegas, NV at links.basis.com/tcreg

For more about the new training format, including course descriptions and to register, go to www.basis.com/training For more about training, including course descriptions and registration, go to www.basis.com/training



Database query performance is always toward the top of the list of priorities for administrators and users of data-driven applications. Tracking down a performance related query issue can be a time consuming and often frustrating process, especially when it involves only a small set of particular database queries. Since we at BASIS use all of our own products for our in-house accounting system, we often think of new ideas for improving the BBj® database in the areas of performance and data validation and as a result, we improved Table Analysis and the Query Analysis tool.

Unlike other commercial RDBMS databases that have an almost infinite number of constraints on what and how the data can be entered into the database, the BASIS database comes with a history of allowing developers to have full programmatic control of what and how they store their data. This flexibility means that BASIS has to offer extra functionality to help the developers shape or enhance their data and descriptions into a more SQL friendly format to achieve optimal access via both native and SQL

access respectively. Table and Query Analysis are two of the features that now give you more information about how to enhance your query performance with very little effort.

Table Analysis





Table Analysis performs an analysis of the tables in a database and gathers information about the tables' contents and structure that allows the SQL optimizer to more accurately determine how each query can be optimized when applied to a table. While the way the SQL engine uses this information has not changed, we made significant improvements in two areas: efficiency in gathering the information, and the addition of data and dictionary validation or "database alerts" during the process.

Improved Processing Speed

Version 12 of BBj and above, includes an improved process for gathering the table analysis information. With this change, customers see the time it takes to analyze large databases cut by several hours. For example, an extremely large database from our partner Audev (www.audev.com) took 11 hours to analyze in version 11.11 of BBj. In the improved version 12, the analysis took only 3 hours, a tremendous gain drastically reducing the load on the machine. This runs in the background and would generally only run once and would only require a re-analysis if there were significant changes to the structure such as the addition or deletion of indexes, or changes to the statistical makeup of a table like doubling the number of records.

Database Alerts


Since Table Analysis iterates over the table data inside the database to gather its statistical data, it made sense to add some data and dictionary validation to this process as well. Further, the Enterprise Manager now indicates the state of each table with an icon next to the table name in the table list. The four states are:

	Table analysis complete - no alerts.
	Informational alert - something BBj-specific about the definition of the table, i.e. how a particular index is limited in use for SQL optimization due to the type of data in the field.
	Warning - a possible problem depending on how the table is used, i.e. a variable length character column defined with one length but contains data longer than the defined length can cause problems when a third party application accesses it.
	Critical alert - a problem in the dictionary definition or the data in the table does not match the definition in the dictionary.



By Jeff Ash
Software Engineer

Figure 1 shows the newly improved Enterprise Manager interface which now displays the status of each table with one of the four icons, as well as an example of a warning alert message. Note that the alert also gives instructions on resolving the issue.

Not only does the Enterprise Manager display alert messages, but it can also automatically fix certain types of alerts. To see this in action, select the tables to auto-fix and click the  button at the bottom of the window. This displays a dialog with a list of the tables and alerts that Enterprise Manager may be able to fix automatically, depending on the underlying issues.

Query Analysis

Query Analysis displays information about the types of WHERE clauses used in queries executed on the database to aid administrators in improving indexing of their tables. It displays the columns included in the WHERE clause as well as whether those columns are indexed or partially indexed. The Query Analysis interface in the Enterprise Manager also has a new look, making it much easier to locate potential performance issues and then resolve those issues quickly and easily.



Note the warning icon  on the items in **Figure 2**. The Column List column shows the list of columns used in the WHERE clause. Since the Optimizable Index column is empty, this indicates there is no index that utilizes that list (or partial list) of columns. The Score indicates the likely number of records that the SQL engine would need to iterate over to return a single value matching the WHERE clause, while the Record Count column shows the total number of records in the file. The Pct. column indicates the percentage of the records in the file that would likely be examined in order to return a single value based on a query using the displayed columns.

Figure 3 shows two other types of icons. The green circle with check mark  indicates there is a primary or unique key/index on the combination of columns, which means it only needs to iterate over a single record to return a value for the query. The other icon, a small grey bullet, indicates that there is an index available for optimization and using the value in the Score column

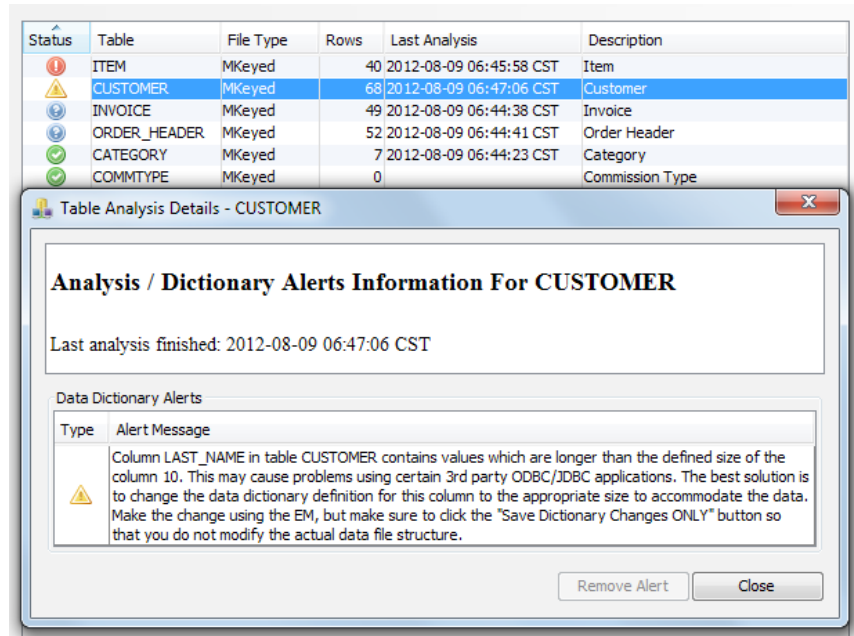


Figure 1. Example of the new Dictionary Alert








Status	Table	Column List	Count	Optimizable Index	Score	Rec. Count	Pct.
	ART13	LINE_CODE	8716		609319	609319	100.00%
	TRM03	END_USER_NBR	5890		10495	10495	100.00%
	ART73	TYPE_OF_SALE	5751		235904	235904	100.00%
	SNM01	PRODUCT_REV	1612		207960	207960	100.00%
	SNT01	PRODUCT_REV	1432		637980	637980	100.00%
	CANADIAN_EXCH	RECEIPT_DATE, AR_CHECK_NBR	1390		27	27	100.00%
	SNM1	ACTIVE_FLAG	1394		207960	207960	100.00%

Figure 2. Updated Query Analysis panel with alert status









Status	Table	Column List	Count	Optimizable Index	Score	Rec. Count	Pct.
	GLM_ACCT	FIRM_ID	14	PRIMARY (0)	24	141	17.02%
	GLM_ACCT	GL_ACCOUNT	15	AO_ACCT_F...	2	808	0.25%
	GLT_TRANSDetail	FIRM_ID, GL_ACCOUNT, TRNS...	134	INDEX0 (0)	1105	658821	0.17%
	ADC_COUNTRY	CNTRY_ID	30	COUNTRY (0)	1	11	9.09%
	ADC_LANGUAGE	LANGUAGE_ID	20	PRIMARY (0)	1	10	10.00%
	ADM_MODULES	ASC_COMP_ID, ASC_PROD_ID	9	COMP_PROD...	1	23	4.35%
	ADM_USER	USER_ID	2	PRIMARY (0)	1	23	4.35%
	ADM_SESSIONS	ED_TABLE_NAME, PER_COLUMN	4	PRIMARY (0)	1	55	1.82%

Figure 3. Updated Query Analysis panel with additional status types

in conjunction with the Record Count allows the administrator to see the optimization level of queries with this combination of columns.

Summary

Performance of an application is very important because it determines how efficiently users can enter and retrieve data, as well as contributing to the overall user experience. Sometimes BASIS can give customers improved performance by refactoring or enhancing our products. However, sometimes it requires the developer or administrator to make some changes to their database structure to gain improvements.

The Table Analysis performance improvements cuts down on the time necessary to analyze the tables while the improvements to the Query Analysis interface make it easier for administrators and developers to locate potential areas for improvement in SQL query performance by better indexing their tables based on user usage of those tables. If you are using Barista, our data dictionary driven RAD tool that makes heavy use of SQL, or you are running any third party SQL tools against your BASIS data, then dramatic performance improvement is just a click away! ■



Looks Better, Runs Faster

GUI and BUI Image Optimizations

BASIS went to great lengths to optimize the launch time, execution, and overall performance of the new BASIS Product Suite Download page, discussed in *The Anatomy of a Web App Makeover: A Case Study* at links.basis.com/12webapp. Another area for application optimization that we explored while reviewing the new BUI download page was that of image optimization – one that developers too often woefully ignore. Image optimization is really ‘low-hanging fruit’ as it does not take much time to review the graphics in an app. Optimizing file sizes can lead to big speed boosts when the app runs in a high latency or tiered architecture. And choosing the best tool for the job can also ensure that images render in the best quality possible.

This article covers a few different ways BBj® developers can optimize their application’s images to achieve quicker display times, make the most of limited bandwidth, and deliver quality output.

Format

Choosing the best image format for the individual graphics is the foremost potential optimization that developers often overlook. Various image formats, such as .png, .jpg, and .gif, employ different compression algorithms that reduce the size of the final image. These compression algorithms vary from format to format, and some are more effective for certain image types. On the flip side, some algorithms do a lousy job on images with certain characteristics, so determining the optimal format can result in huge savings. Colors are one such characteristic. A photo with millions of colors, for example, will usually compress the best using the .jpg image format. On the other hand, a logo with just a few colors will probably compress best using a .gif or .png format and won’t include unsightly compression artifacts that a .jpg format could introduce.

File Size

By way of example, you saved a large photograph as a .gif that weighs in at 407 KB, taking 5 seconds to download on a 1Mbps cable modem Internet connection. In addition to taking a long time to download, .gif files are palette-based and reduce the number of colors to a set of 256-fixed colors in a color table, which results in a dithered image with reduced quality. In contrast, saving the image as a .jpg file would result in a 104 KB file – almost one quarter of the size of the .gif image! Not only will this file download much more quickly, but you have the ability to control the level of compression to make a good compromise between image quality and file size.

To drive the point home, saving the exact same image as a lossless 24-bit .png image with alpha channel support (which will not even be used), would result in a 1.1MB file that takes 13 seconds to download! Selecting the optimum format for

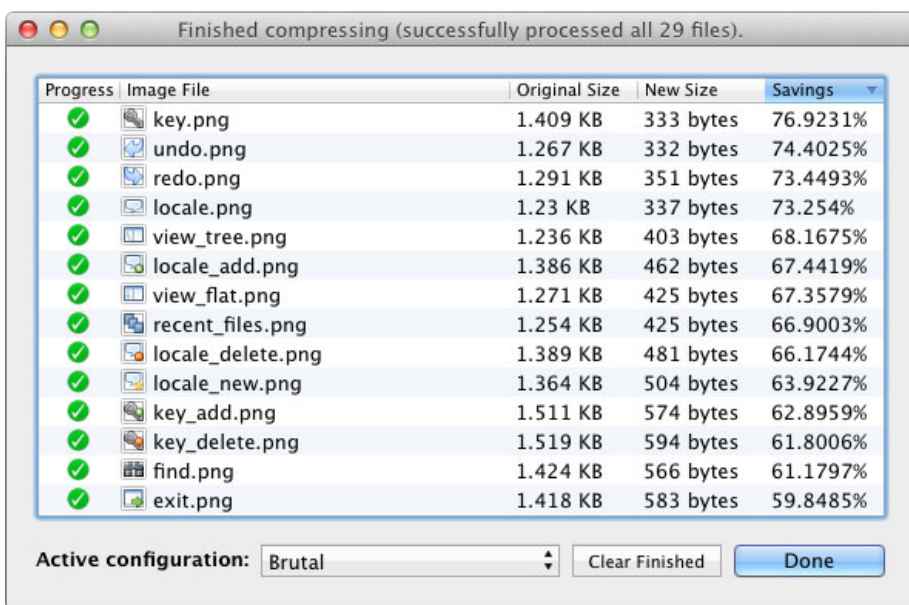


By Nick Decker
Engineering
Supervisor

the size and type of image is occasionally a trial and error process, but it has the potential to yield huge rewards by dramatically reducing disk space and transfer time.

Crushing

Here at BASIS, we capitalized on our knowledge of ‘crushing’ to further optimize our [BUI Mortgage demo](#) when we changed the vertically-tiled background image from a .jpg format to an 8-bit indexed .png format. We then used a third party utility like [pngcrush](#) to strip out unnecessary metadata from the image and decreased the size of the background image by 400% – from 7.6 KB to 1.9 KB. Running several .png toolbar images used in other BASIS utilities also resulted in significant savings as shown in **Figure 1**. The new, smaller images not only take up less storage space on the disk, but more importantly, the smaller file size results in faster transfers from the server to the client. Depending on the number of images used in an application, this can have a measurable impact and reduce the application load time.



Progress	Image File	Original Size	New Size	Savings
✓	key.png	1.409 KB	333 bytes	76.9231%
✓	undo.png	1.267 KB	332 bytes	74.4025%
✓	redo.png	1.291 KB	351 bytes	73.4493%
✓	locale.png	1.23 KB	337 bytes	73.254%
✓	view_tree.png	1.236 KB	403 bytes	68.1675%
✓	locale_add.png	1.386 KB	462 bytes	67.4419%
✓	view_flat.png	1.271 KB	425 bytes	67.3579%
✓	recent_files.png	1.254 KB	425 bytes	66.9003%
✓	locale_delete.png	1.389 KB	481 bytes	66.1744%
✓	locale_new.png	1.364 KB	504 bytes	63.9227%
✓	key_add.png	1.511 KB	574 bytes	62.8959%
✓	key_delete.png	1.519 KB	594 bytes	61.8006%
✓	find.png	1.424 KB	566 bytes	61.1797%
✓	exit.png	1.418 KB	583 bytes	59.8485%

Active configuration: Brutal [Clear Finished] [Done]

Figure 1. Compressing the images used by the BASIS utilities saves time and space

Uniting Images for Network Optimization

Applications vary in the number of images they employ, but it is fairly common for them to use dozens of images for menu items and toolbuttons when the application is sufficiently complex. Sending these images over the wire to the client may seem innocuous since they are usually less than 1 KB in size, but often times the number of HTTP requests required to transfer the images to the client is the crucial aspect and limiting factor. Reducing the number of HTTP requests is one of the foremost methods webmasters use to speed up their websites, making pages load faster and minimizing the effect of latency. Images are prime candidates for this form of optimization, as combining multiple discrete images into a single image file slashes the number of HTTP requests from dozens down to one. Webmasters normally rely on image maps or CSS sprites to combine image files, and BBx® developers can reap the same rewards by using an ImageList. ImageLists are typically used to set an image in a TabControl or Grid, but you can use them with any control that offers a `setImage()` method.

For example, we modified the [Resource Bundle Editor](#) to use an ImageList for the menu item, toolbutton, and button images. The application previously used 28 distinct image files, so with a little help from a CSS Sprite tool, we combined them into a single image comprised of the concatenated images as shown in **Figure 2**.



Figure 2. The result of combining multiple images into a single ImageList

Analyzing a BUI instance of the application in [Chrome's Developer Tools](#) confirmed that the number of HTTP requests dropped from 28 down to just 1. Using the ImageList will not only drastically reduce the amount of communication between the client and server, but the resultant ImageList file will almost always compress more than the individual images (due to factors like discarding redundant header information). Our example showed that our changes not only eliminated dozens of HTTP requests, but also reduced the amount of data sent – from a combined size of 27 KB for the 28 separate images to just 10 KB for the ImageList. Combining your application's images in this fashion optimizes network performance, reduces the effect of latency, and best of all, benefits traditional thin client and BUI apps alike.

CSS Image Optimizations for BUI

If your BUI app uses multiple images as part of its CSS styling, CSS Sprites are the logical way to optimize them to reduce HTTP requests. The webpage css-tricks.com/css-sprites has a good overview of sprites with an example that shows how combining images into a single sprite reduces the number of HTTP requests from 10 down to 1 and reduces the total size of the images from 20.5 KB down to 13 KB. Even better, as images sometimes download on a 'lazy' or 'as-needed' basis, it is far better aesthetically to use a single image.

When the browser displays a control defined with separate images for the various states (normal, hovered, and active), it loads each image the first time it is needed. So loading the image files for the hovered and active states occurs when the user first interacts with the control. This takes a fraction of a second and the control flashes as the original image is removed and the new image loads and displays. When using a single sprite image, the entire image is loaded at the start and already cached by the browser when it comes time to show the other controls states so the transition occurs instantly without any visual disruption. Various free and commercial sprite editors exist, making this potentially time-consuming and exacting task a piece of cake. Some editors, such as the one shown in

Figure 3, not only take care of combining images and optimizing layout, but also write all of the CSS code for you!

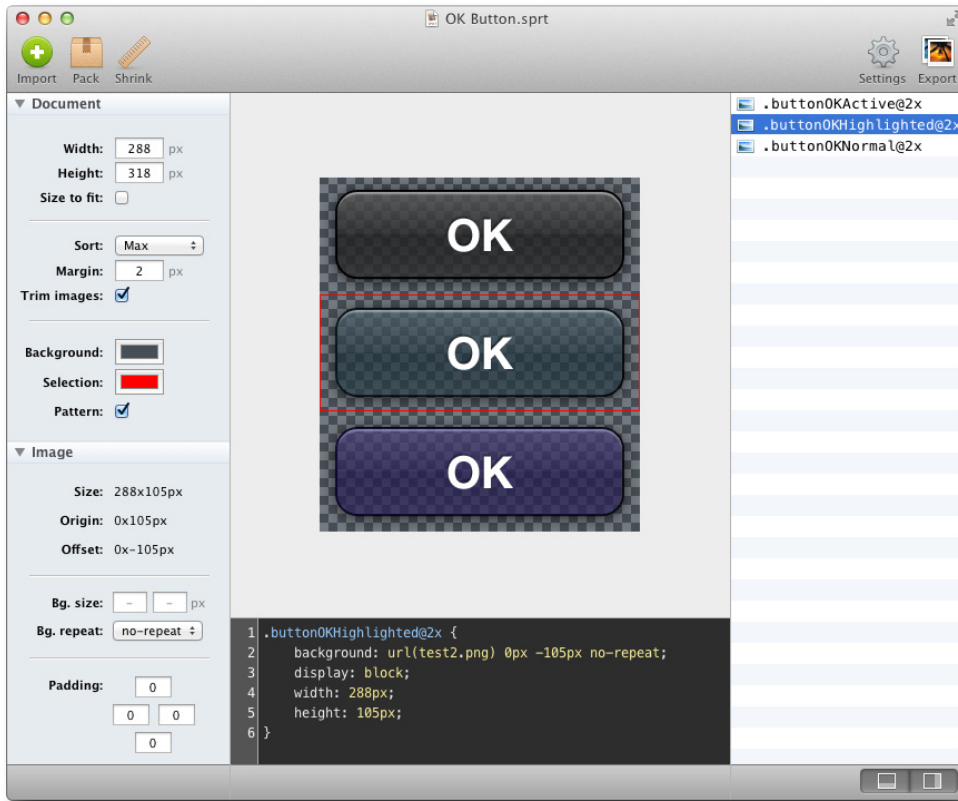


Figure 3. A sprite editor that combines images and writes the supporting CSS code

Optimizing for Retina Displays

“Can BUI apps take advantage of one of the new ‘Retina Displays?’” is a question that has started to pop up more frequently lately. With the initial release of the iPhone 4, and with the recent releases of the latest iPad and MacBook Pros, Apple has been touting their Retina Displays. The idea is that these devices utilize high pixel density, meaning that they pack a huge amount of pixels into the device’s screen. The result is a pixel density of more than 300 pixels per inch, even better than some of the early laser printers. The pixels are so small that they cannot be individually discerned when holding the device at a reasonable reading distance, meaning that text and graphics are ultra-sharp. Many Android-based phones also utilize high pixel density and their [screen support API](#) covers devices with various pixel densities and defines concepts such as density independence.

So what does this all mean to BBj BUI programmers – are their apps going to be able to play in the high-resolution game? The answer is a resounding “Yes!” and in most cases, programmers will not have to do anything special or make any code changes. Our aforementioned [BUI Mortgage demo](#) serves as a good example of this as seen in **Figure 4**, which displays a small section of the app running on the new iPad with its whopping 2048x1536 pixel screen resolution.

The titles, labels, and input controls are all razor-sharp and match the high-resolution native iPad apps because most BBjControls and text fall into the ‘vector’ category. Vector means that they can scale well without degrading in quality. However, developer supplied images used in custom CSS fall into the ‘raster’ category, which means that they degrade dramatically when enlarged.



Figure 4. High resolution BUI app running on a Retina Display iPad

Optimizing BUI CSS Images for Retina Displays

Optimizing custom images for a high pixel density display is possible given CSS' media query capability. The CSS in **Figure 5** specifies a background image (**my-image.png**) for regular displays. But when the client browser is using a pixel-doubled display such as the iPhone or iPad, a different image (**my-image@2x.png**) is used instead. This version of the image has four times the resolution, twice as many pixels in both the X and Y direction.

```
.my-selector {
  background-image: url(my-image.png);
  height: 100px;
  width: 200px;
}
@media only screen and (-moz-min-device-pixel-ratio: 2),
  only screen and (-o-min-device-pixel-ratio: 2/1),
  only screen and (-webkit-min-device-pixel-ratio: 2),
  only screen and (min-device-pixel-ratio: 2) {
  .my-selector {
    background-image: url(my-image@2x.png);
    background-size: 200px 100px;
  }
}
```

Figure 5. Defining regular and high-resolution images for Retina Displays

The **@media** portion of the CSS file allows developers to specify selectors that the client browser will use when viewed on a computer or mobile device with a pixel-doubled screen. The redefinition of **.my-selector** with the high-resolution image will take precedence over the initial definition due to CSS' cascading order. Because they both have the same weight, origin, and specificity, the last definition 'wins' and is the one that the browser will use. The final trick involves setting the **background-size** property for the selector to the same 200x100 pixels specified in the original definition. In essence, we are directing the browser to display the pixel-doubled image to our 'preferred' size of 200x100 pixels. If we omitted the **background-size** property, the image would display twice as wide and tall as the normal resolution image. By specifying the **background-size** property, we force it to use the native display pixels to squeeze it into the same screen real estate.

We used this same technique for the **BUI Tip Calculator** demo targeted for the iPhone. The app uses a custom image defined in a CSS file to display an interactive service rating as a series of stars. Simply tap on a star to define the level of service you received and the tip adjusts automatically. The "How was the service?" label is a **BBjStaticText** control, so it looks perfect on

a pixel-doubled display without any extra work on our part. However, because the image used for the star rating system was defined as a regular image in a custom CSS file, it did not scale well and appeared blurry when viewed on the iPhone 4 and above, as shown in **Figure 6**.

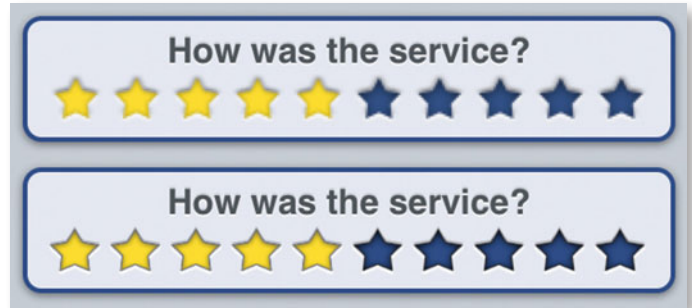


Figure 6. Comparing the non-optimized image (top) and retina-optimized image (bottom)

To make the application look great on the Retina Display iPhones, we created a version of the star image that was pixel-doubled – twice as many pixels for both the width and the height. This ensures that the image used for the rating system will be of the highest quality for every device.

Summary

As computer applications have matured and migrated from CUI to GUI and now to BUI, images have become an integral part of most of these applications. In addition to adding aesthetic value, when used wisely they improve usability and provide interactive feedback. Despite their importance, they are sometimes treated as an afterthought and even though developers typically profile their application for performance, they may overlook the importance of optimizing their application images as well.

Taking the time to analyze and compress images is an important opportunity to improve the launch speed of an application while preserving image quality. Fortunately, optimizing images is fairly easy and quick, especially with some of the advanced compression tools available. You can now accomplish this task yourself instead of delegating it to a dedicated graphic artist. BUI applications look great out-of-the-box on the new Retina and pixel-doubled displays without resorting to custom code. If your BUI app happens to utilize images in a custom CSS app, with a little CSS kung fu, you can make image degradation a distant memory and produce a fantastic looking app! ■



- Try out these BUI demos
 - Mortgage Amortization Schedule at links.basis.com/buimortgage
 - Tip Calculator at links.basis.com/buitip
 - More at links.basis.com/buidemos
- For a more in-depth discussion of image formats and criteria to help determine the best image formats, review Image File Formats at en.wikipedia.org/wiki/Image_formats
- Check out these tools
 - Pngcrush at bit.ly/4uyudU
 - Chrome Developer Tools at bit.ly/HNgdC0
 - Android Screen Support at bit.ly/mithf



In today's world, we see Web Services (WS) in action everywhere from online shopping and shipping trackers to mapping and geolocation tools. The methods used to create WS are just as broad and varied as the tools that use WS. However, some of the protocols defined by the WS- documents seem to exist only for Oracle to sell consulting contracts. One can easily drown in the alphabet soup of UDDI, WSDL, and SOAP. Learning SOAP (Simple Object Access Protocol) reveals there is nothing simple about it. Add to that the fact that Java and .net are incompatible in subtle ways, which creates extra complexity. REpresentational State Transfer (REST) eliminates much of the formality imposed by a WSDL-style WS.

How does REST avoid complexity? First, REST is stateless. Statelessness alone won't make WS simple. But, leveraging the Internet's most popular stateless

protocol, [Hypertext Transfer Protocol](#) (HTTP), simplifies WS even more. Statelessness and HTTP let us take advantage of many existing tools. The whole spectrum of Internet infrastructure helps us implement REST-based WS. From Java libraries that provide valid HTTP sessions to caching web proxies, all of the infrastructure on the Web works for us.

What exactly is a stateless protocol? Imagine a stateless bank teller. A conversation might go something like this:

Jason: "Excuse me, ma'am, what is my balance?"

Teller: "I'm sorry, sir, for what account?"

Jason: "Of course. What is the balance of account 1234?"

Teller: "I'm sorry, who is the account holder for account 1234?"

Jason: "What is the balance of account 1234 held by Jason?"

Teller: "I'm sorry, what is the password for account 1234 held by Jason?"

Jason: "What is the balance of account 1234 held by Jason with password GetItAlready"

Teller: "The balance is 37 cents."

Stateless services require all the information about a request up front. Providing partial information will not return a valid reply so each request sends whatever the server might need in order to get a valid reply. The bank itself is stateful. They, of course, need to remember all of their customers and balances. Each request to the bank is a self-contained package, requiring no extra state. Statelessness simplifies the client and server code greatly. In fact, it's so simple we can use HTTP instead of writing our own SOAP-like system.

How, exactly, is information sent to the web server? A web browser uses HTTP to talk to a web server. Every click of a link on a web page sends an HTTP request to the server and returns an HTTP response back to the client. The protocol is a little bit too complicated to just type into a telnet session, but it is not nearly as hard as the alphabet soup of WSDL services. There are four key HTTP methods used in REST-based WS: GET, PUT, DELETE, and POST.



By Jason Foutz
Software Programmer

Every query asked of our fictional bank teller was a GET request. Requesting balance information GETs information about the bank account, a specific resource. In SQL, a SELECT statement is like the GET request. GET requests are made visible on many websites. At www.google.com, searching for something adds on many parameters about the request. It might look something like www.google.com/search?query=something. The parameters vary depending on the exact query, but the arguments display inside the address bar of the browser.

A PUT request replaces existing state on the server. Our fictional bank teller would be able to change a password with a valid PUT request. A PUT request is similar to the SQL UPDATE statement. Furthermore, PUT requests should be idempotent. Whether using PUT to change the password to "hello" once or 100 times, the password should be "hello" after every request. Intentionally, or accidentally, sending the request multiple times shall have no effect.

A DELETE request does just what it sounds like it does. DELETE removes state from the server. HTTP DELETE works like an SQL DELETE statement. Because it changes state, DELETE should also be idempotent. For example, deleting a specific account multiple times has no effect, no additional accounts are deleted.

The final HTTP method, POST, creates resources. POST is similar to the SQL INSERT statement. Every POST, creates a new resource, so it can not be idempotent. Depositing a dollar into an account POSTs that dollar to the teller. Every single dollar deposited has an effect, to create a resource on the server.

So how would we actually implement a REST Web Service? The BBj® Servlet API provides full access to HTTP. This introduction just scratches the surface of what is possible with BBjServlet. The

BBj Servlet Overview in the BASIS online docs at links.basis.com/servlet covers creating and handling HTTP requests using BBj Services. In the following examples, `getMethod` is the workhorse for determining what kind of request the user makes. A lot more functionality is available in `BBjHttpRequest` and `BBjHttpResponse` in the online help at links.basis.com/basishelp.

When the server receives an HTTP request, the `getMethod()` determines which HTTP method the client used. You can implement each method GET, PUT, DELETE or POST to handle the client's requests. Since not every WS requires every method, just omit unnecessary methods. Clients must provide all the information possibly needed up front because HTTP is stateless.

Code for transaction processing might look something like this:

```
method public void transactions(BBjServletEvent p_event!)
  request! = p_event!.getHttpRequest()
  method$ = request!.getMethod()
  if "GET" = method$ then
    #showTransactions()
  endif
  if "POST" = method$ then
    #createTransaction(request!)
  endif
methodend
```

This type of transaction cannot be changed or removed so there is no need to implement PUT or DELETE.

A REST-based WS can be consumed with the `http-commons` library. Code for processing a request might look like this:

```
url$ = "http://server:8888/servlet/example"
client! = new org.apache.commons.httpclient.HttpClient()
request! = new org.apache.commons.httpclient.methods.GetMethod(url$)
client!.executeMethod(request!)
```

REST services requires only HTTP from their clients. This makes it exceptionally easy to consume services from many different languages. BBj and Java, as we saw, may use the `http-commons` library. C# ships with an `HttpClient` class right in the .net runtime. While PHP has nothing built in, many third party http client libraries are available. HTTP is so widely used, command line libraries such as `curl` or `wget` could be used from any language's version of SCALL. C programmers might do an `exec` of `curl` to retrieve their WS results. Libraries for interacting as an HTTP client are widespread, virtually every platform and language can call a WS.

Summary

While WS provide unlimited complications, REST WS limit that complexity in two key ways. REST is stateless, requiring clients to provide all the information a server might need for every request. REST leverages HTTP, simplifying the stateless communication to the server. REST is a powerful tool for deploying WS without the risk of drowning in alphabet soup. So rest easy, Keep It Simple and Stateless, and 'KISS' WSDL goodbye! ■



- Review `getMethod` in the online BASIS documentation at links.basis.com/getmethod
- See also
 - Detailed examples of using the Java library at bit.ly/euhVG
 - Sample API of accessing another's Web services at bit.ly/Jlywat
 - Vast list of REST Web Services at bit.ly/9XXsrF



BASIS Survived the Amazon Outage

With all of the hype that is swirling around the cloud these days, one tends to forget that cloud deployments suffer from many of the same risks as the server rooms that we are familiar with, and that many of us have to manage each day. While it is true that with all of the backup power supplies and forty odd thousand servers in each facility, power outages are seldom and there is even less chance that another server is not available any time you need it. However, there is still that slight risk that something catastrophic can occur that will make the entire region come to a screeching halt!

As it turns out, this risk turned into a reality twice on the East Coast since BASIS moved their servers into the cloud. The first time, we had lulled ourselves into a false sense of security that often comes with your first cloud deployment, so we were offline for several hours like the majority of the Amazon customers in the region. Once the cloud came back online, BASIS implemented the necessary features to prevent such a calamity from happening again. Fortunately, it was not a difficult challenge to solve. With a small change to the configuration and a few cents more per hour, we enabled the Multi-AZ

RDS system for www.basis.com and www.addonsoftware.com thinking that we would probably never need it, but we wanted to be safe.

A little more than a year later, the East Coast region failed again and numerous big name online businesses were down again; several companies suffered enough losses that they left Amazon entirely. Fortunately, neither BASIS nor AddonSoftware® suffered an outage at all. Even though our primary availability zone failed, the secondary zone took over automatically and without a hitch. This experience made us a firm believer of the benefits of the Multi-AZ service:

"In the event of planned database maintenance, DB Instance failure, or an Availability Zone failure, Amazon RDS will automatically failover to the up-to-date standby so that database operations can resume quickly without administrative intervention." Amazon Web Services, amzn.to/RDVowj

In addition to keeping our websites up 100% of the time, we also want to keep our BUI demos and BUI-powered b-commerce® and product download pages functioning 24/7 as efficiently and as effectively as possible. To this end, we utilized a different feature that Amazon offers, Route 53 DNS Web Service (aws.amazon.com/route53). While this DNS works like almost any other DNS, it gives us some features that other DNS systems do not. First, it allows us to reduce the time-to-live (TTL) down to only 5 minutes from the usual minimum of 30 minutes. The result is that

our failover from one server to another, or one region to another, will be less than 5 minutes of dead air, no matter how catastrophic the failure is. Another significant reason we use Route 53 DNS for www.poweredbybbj.com is to employ their lowest latency-based routing to provide geo-aware DNS service to our customers worldwide. Now Route 53 will silently direct customers to the lowest latency server in their region, in Europe they are routed to Ireland, US East Coast customers are routed to Virginia and US West Coast customers are routed to California or Oregon. Using the BASIS BBj® replication service to replicate the programs, configuration, and data to all of these machines means that we can deliver optimal performance around the world without adding to our process management overhead. We still just maintain the same single source machine and replication keeps all of the target machines up-to-date without any human intervention.

In order for BASIS to continue with Nico's "belt and braces" redundancy desire (aka "belt and suspenders" in my part of the world), we utilize the new scheduling feature in Enterprise Manager (EM) to pause the replication in the middle of the night, just long enough to run a traditional incremental backup to an Amazon storage bucket. Then we use another EM-scheduled task to rsync the production Amazon drive to another Amazon drive, giving BASIS two backups in addition to the replicated databases. Therefore, if a catastrophic failure ever occurs in one or more regions, we have replicated databases that we can fail



Dr. Kevin W. King
President & CIO

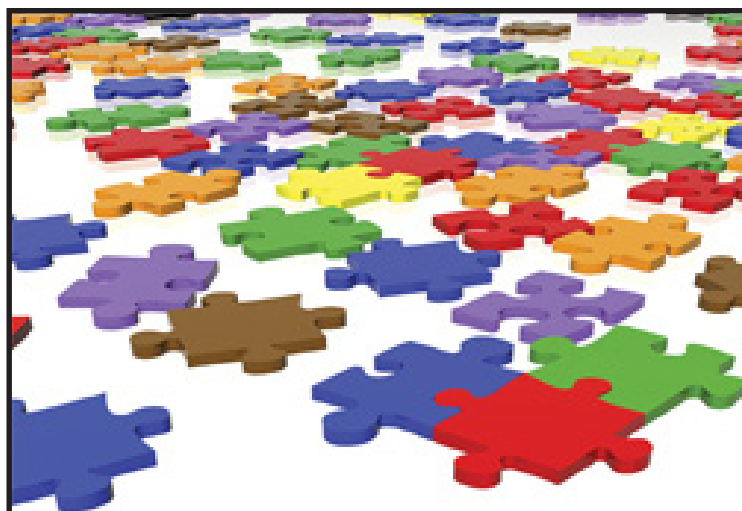
over onto, which are located on three different continents in less than five minutes. We still have the traditional midnight backups, should we ever need them, like our braces or suspenders. The combinations of configurations that are possible using the latest BBJ toolset is almost endless. We are confident that one or more of these options will benefit all of your needs.

Finally, you might ask, *"How can anyone afford all of this hardware in all of these locations, even if the human maintenance cost does not increase?"* Well, [Moore's law](#) and market forces continue to drive costs down and technology up. The Amazon price of all of our replication servers have dropped to 1/6th of the on-demand servers, so we run replication servers in all of our high density customer regions for only 1-3 cents per hour and the storage price is calculated at pennies per GB per month. In order to keep our human costs low and our reliability high, we have put all of our replication, archive, test, build, and demo machines on spot instances, mostly controlled by Amazon Auto Scalars (aws.amazon.com/autoscaling). In response to trouble that might arise in any region, a new instance comes up in a different region and the removal of the DNS entry in Route 53 for the dying machine occurs automatically, so that BASIS customers do not experience a break in service and BASIS employees don't experience a break in their slumber.

While BASIS benefits from all of these cloud features, you might wonder how all of this benefits you, beyond having your language provider always accessible and online. Well, not only does having your language provider soaring high in the cloud give you a sense of comfort and security in that BASIS is prepared to handle calamities of any nature, it also gives you confidence that the language has built-in features and the capacity for you and your customers to move into the cloud infrastructure. Utilizing all of these features, we are able to add scalability, redundancy, and rapid recoverability for our AddonSoftware application in the cloud called AddonSoftware Cloud Service, or Addon Cloud for short. Utilizing Amazon's AutoScaler and Route 53 DNS along with BASIS' Data Replication and Scheduler, the Addon Cloud offering delivers geo-aware DNS for reduced latency, automatic server redundancy/replacement, and rapid recovery or failover functionality for any regional cloud interruptions. If you are looking for a robust affordable ERP solution, you would be hard pressed to find one that is more architecturally reliable and redundant than the Addon Cloud solution. ■



For more information, see *Are You Prepared for Cloud Failure?* at links.basis.com/12cloud



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/S 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



ASCI Partakes RADically From the Barista Cup

Soon after BASIS introduced the Barista® Application Framework to the BASIS product suite, ASCI of Miami was quick to recognize the benefits of implementing the RAD tool's built-in feature-function into their own solution, and getting SQL access to their data. Enabling ASCI to add powerful features beneficial to their end-users that ASCI did not have to develop made this an easy decision. With a broad offering of applications for their vertical market, ASCI strategically planned to implement their application migration in two phases.

Phase 1

In the first phase, ASCI introduced improved navigation (see **Figure 1**) through an all-new multiple document interface (MDI) with enhanced search/sort capability, new report output options, tables/forms creation, and role-based security with an audit trail – all built-in features of Barista. More specifically, users immediately enjoyed these significant time savers and new security measures:

- **Navigate through the more user-friendly and structured menus of Barista**, providing a more modern and easier, more consistent, user experience within the programs.
- **Open multiple windows simultaneously using the new MDI**, allowing quick and efficient navigation through several open ASCI windows and in so doing, increase productivity.
- **Search and sort by any field in an inquiry window** or create custom inquiries, giving the user access to the data that empowers them to make informed business decisions.
- **Output reports in PDF, XML, XLS, HTML, CSV and Google doc formats**, equipping users with the ability to create their own reports and provide them the ability to archive the reports in the format that best suits their needs – otherwise only available from expensive third party packages.

- **Have tables and forms created for them with the desired look-and-feel**, allowing customization “just the way you like it.”
- **Implement role-based security with an audit trail, right down to the field level**, offering security and the confidence that only authorized users can make changes with an automatic record of the database changes.

As of this writing, ASCI has already completed the first few Phase 1 installations of ASCI - Powered by Barista (**Figure 2**) and is receiving very positive feedback. ASCI Programming Manager Alex Castellanos says,

“We are confident that Barista will meet our technology requirements to allow us to meet our customer needs well into the future. We love how the user interface works and the way Barista moves...it is very fast. We are glad we made this move and are anxious to upgrade all our customers to the new ASCI - Powered by Barista solution.”

Phase 2

In this next phase, currently in process, ASCI is extending the Phase 1 features across all their remaining ASCI modules including Export Procedures, Warehouse Procedures, AMS, and Accounting. ASCI has committed several programmers as well as Quality Control personnel to this project and are making great progress in the conversion. In addition, ASCI is converting their Data Entry programs to take full advantage of Barista. Barista provides ASCI the ability to develop their user interfaces much more quickly and efficiently.

Since ASCI can now more quickly customize and enhance their customers' programs, ASCI can provide better, faster, and more affordable customer service.

Castellanos describes their experience learning Barista,

“We did not find the learning curve so difficult. Most of our staff had experience working with this kind of development tool. Also, the online training sections and online tutorials helped us to get familiar with the basics of Barista.”

Castellanos sees many benefits that will carry them far into the future.

“I see many things that will result in a better product for our clients...rapid software development, standard programming routines and callpoints, and how easy it is to create new forms and applications, and to organize and maintain our database system.”



By Susan Darling
Technical/Marketing Writer

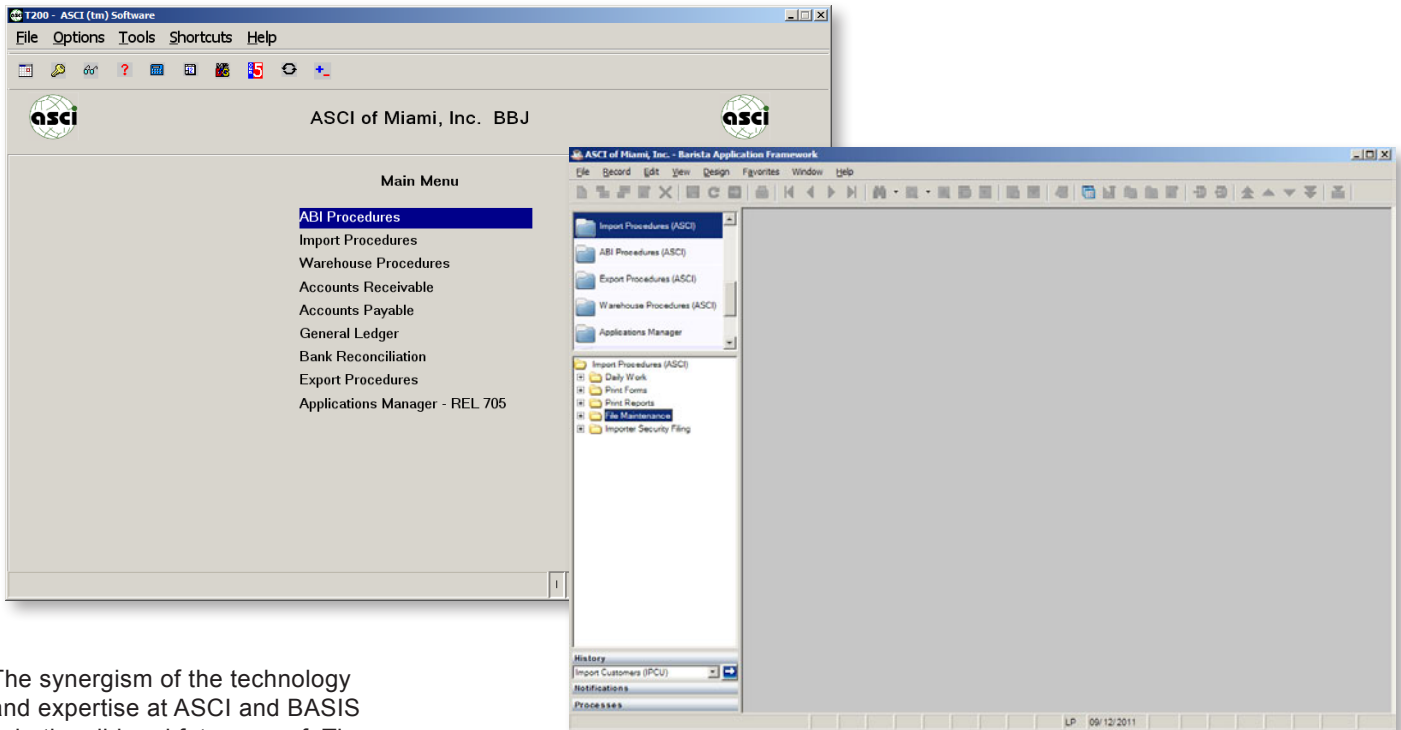


Figure 1. The main menu before (left) and after (right) in the new ASCII - Powered by Barista

The synergism of the technology and expertise at ASCII and BASIS is both solid and future-proof. The ever-changing technology landscape presents ongoing challenges and opportunities with such advances as the cloud and mobile computing. Barista and the underlying language – Java-powered BBj® – deliver a broad eco-system for deployment on a variety of popular operating systems and browsers. Barista applications run on a multitude of platforms and operating systems; equally well with a single operating system or with a mix of Microsoft Windows, Linux/UNIX, and Mac OS/X. These technology components serve as the foundation for modern business application solutions, fully cloud-enabled and mobile capable, all from a single code base in the Java-made-simple world of BBj.

The future for ASCII is secure and bright. Follow their example and RADically change your GUI application with Barista's out-the-box new features and functions. More than just a RAD tool, Barista's framework delivers building block components built into the product. ■

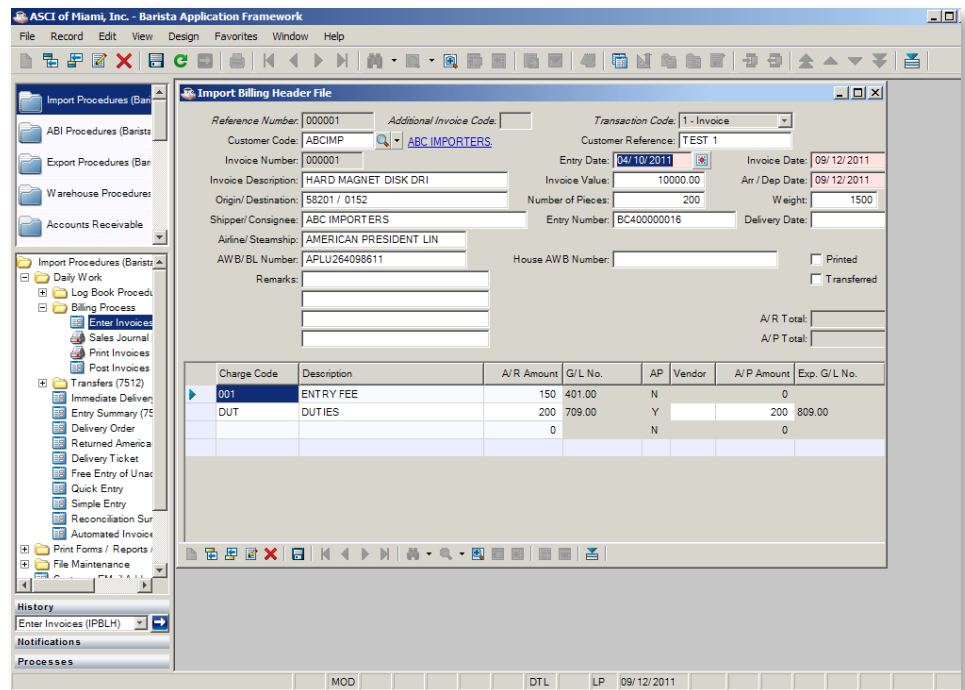
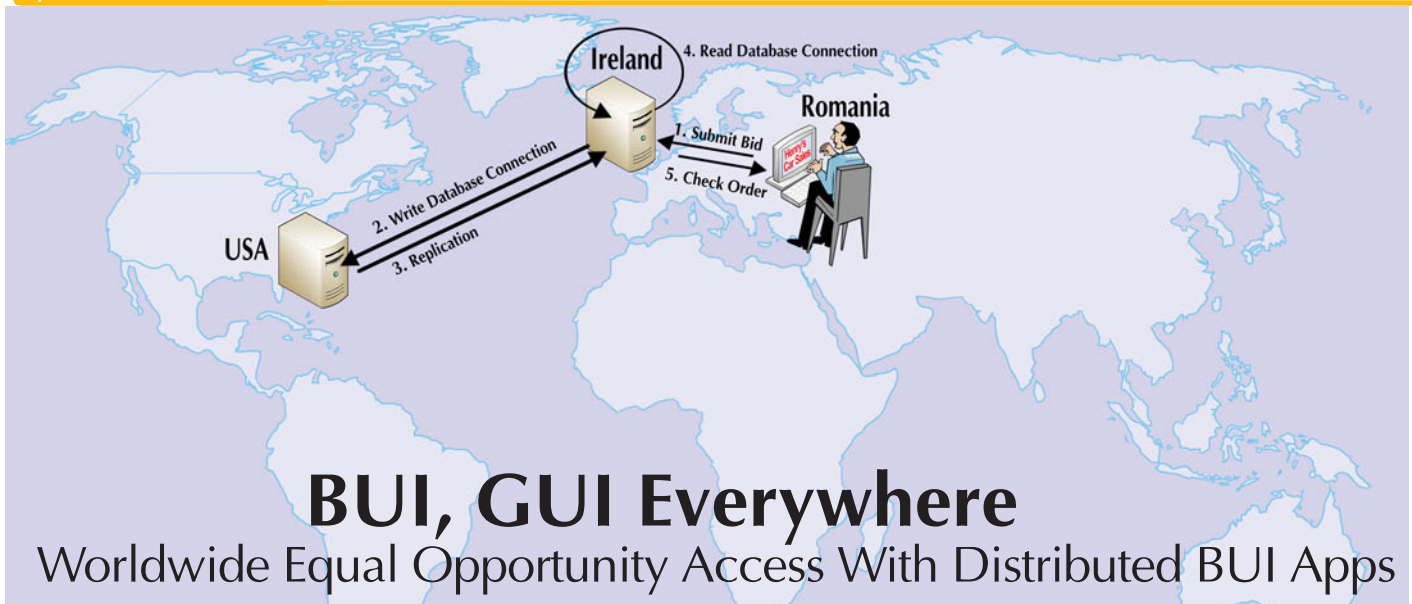


Figure 2. Billing in the new ASCII - Powered by Barista

ASCII of Miami (www.asciomi.com) is a developer and distributor of a complete software solution for U.S. Customs brokers, importers, freight forwarders, and other parties involved in cross-border trade. ASCII is aggressively adopting BASIS' best-of-breed technology to catapult their customers onto their newest state-of-the-art solution. Integrating new features into their original Visual PRO/5® architected software solution, ASCII is taking a giant leap forward. Over 300 satisfied customers currently run the ASCII Software System across the U.S. and in Puerto Rico.





Henry, a developer on the East Coast of the United States, soon got his GUI program running on the cloud server in BUI. Anyone in the world could now navigate to the company page and browse the online database of new cars! His program offered 360° exterior, and interior views of each automobile, allowing the user to choose colors, upholstery, different spoilers, wheels, and other customizations. Customers could build the auto of their dreams, submit it to the dealer, negotiate the price, and have it delivered without ever setting foot in the showroom, all online for any of the dealer's worldwide locations.

But, there were already problems. Henry got reports that the program took two seconds to load for customers in North America, ten seconds to load in Europe, and fifteen seconds to load in Asia. Fifteen seconds is a long time to wait for a page to load. But wasn't this to be expected? After all, his server was in the US; the further away a customer is from the server, the longer the program takes to load.

Henry thought, "Surely, there's a way around this," and actually, there is!

The solution to providing consistently quick access to a BUI app anywhere in the world is to set it up as a distributed application using readily available tools from BASIS and Amazon. While Amazon offers tools for reaching multiple machines with the same address, BBj® offers tools for replicating programs and data across several machines. In this article, we will discuss Amazon geo-aware DNS (Domain Name System) addresses, and BBj File/Directory-Database Replication, and how they can be combined to make Henry's auto sales application run equally well and transparently from anywhere in the world.

The Primary Tools

The two major tools for creating a distributed BUI application are Amazon's geo-aware DNS addresses and BBj File/Directory-Database Replication. Geo-aware DNS addresses allow one DNS to correlate to several IP addresses, where each address is located in a different region. BBj Replication maintains up-to-the-minute copies of programs and data between machines in all of the regions. Let's take a more detailed look at these tools.



By Shaun Haney
Quality Assurance
Engineer

Geo-aware DNS Addresses

One major feature of cloud computing is being able to provide content to users from a server close to them. In the past, you might provide North American users a .com address, British customers a .co.uk address, and German customers a .de address, etc. It would be great if your customers did not need to know what address was near them and you could just use the same web site address all over the world. Geo-aware addresses provide exactly this. Using Amazon's Route 53 configuration tool, you can specify a single DNS name, assign it an IP address, and then specify a region of the world where the DNS belongs. To add more locations, simply add more DNS records with IP addresses and specify a different region for each address. For example, North American customers will go to the IP address of a server on the East Coast or West Coast, while European customers will go to an address in Ireland, South American customers will go to a server in Sao Paulo, and Asian customers will go to an address in Japan. All of those customers will type the same website name in the address bar of their web browser and Amazon will reroute them to the correct machine behind the scenes, without the customer having to do anything.

BBj Replication

BBj replication is invaluable anywhere redundancy is needed. Replicated data is not only a quickly available up-to-the minute backup, but is also a read-only copy of the original data accessible in real time. BBj replication is not just for databases, but for any file that needs a copy maintained on one or more remote machine(s). Replicated BBj programs can run on the replication target machine, but any changes to the programs should be made only on the source machine.

Supplementary Tools

Other important tools include AWS EC2 (Amazon Web Services Elastic Compute Cloud) and BBj databases and files. Amazon's EC2 framework is what allows Cloud developers and administrators to create, configure, upgrade, start, and stop cloud machines. A complete discussion of EC2 is beyond the scope of this article, but the reader should be aware that EC2 is the underlying framework for running cloud servers.

BBj databases and files are also critical to any non-trivial BUI application. Rather than discussing data structures in any detail, it is more important to know that an application's data will replicate along with its source files, and that the "genuine" data will reside on the source machine while a read-only copy will reside on each target machine. The BUI application design should include two connections for each set of data: a "read" connection that accesses the local data and a "write" connection for the "genuine" source data.

The Architecture

Now that Henry knows a distributed BUI application is possible and what tools he needs for the process, what does a distributed BUI application's configuration look like?

A distributed BUI application consists of one source server and any number of target servers such as shown in **Figure 1**. The source server and target servers all have an install of BBj, the BUI application source code, and the application's data. Each server has a permanent IP address. All of the IP addresses are associated with a single geo-aware DNS Name. On each machine, the BUI application is identical and has two connections for each dataset used: a read

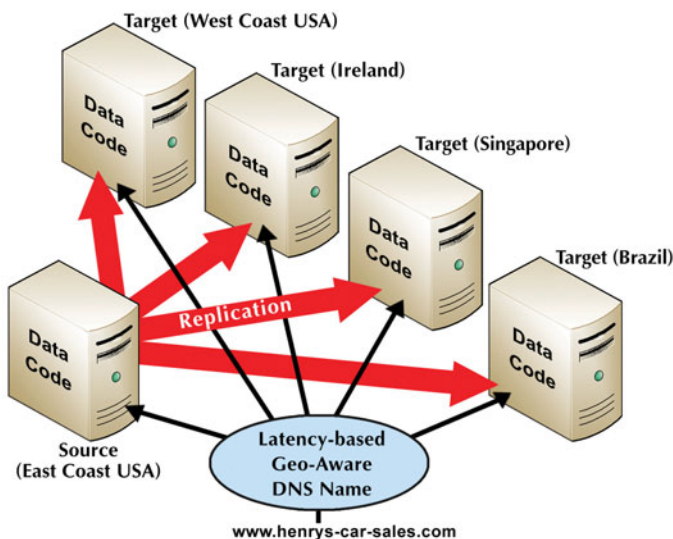


Figure 1. Sample configuration of a distributed BUI application

connection and a write connection. The read connection goes to localhost so that reads occur on the local copy of the data. The write dataset connection goes to an unpublished name for the source server so that all writes are on the source's copy of the data. Meanwhile, replication maintains up-to-the-minute copies of the data, BUI source files, and any relevant configuration files on all of the target machines. In a distributed BUI application configuration, if one makes changes to data, source code, or the application configuration on the source server, replication automatically propagates those changes to the target machines.

After planning the architecture, Henry put his nose to the grindstone. In the course of a week, he sets up servers in California, Virginia, Ireland, Brazil, and Singapore. He chose Virginia for his source server because, even though the source server could be anywhere, it made the most sense to have it in the same region as those in his company who will maintain it.

Now with servers in so many regions, Henry knows that customers will have faster load times for his BUI car sales application and as a huge bonus, he now has several redundant copies of his application and data in case of a failure.

How it Works

With so many servers relatively close to almost any customer in the world and application load times much faster, how exactly does using the BUI application work? Recall that Henry's main server is in Virginia.

While Henry is fast asleep in the U.S., Ivan, a customer in Romania, decides it's time to buy a new car. Ivan simply types www.henrys-car-sales.com into his favorite browser's address bar – the exact same address any other customer in the world would type – to connect him to the closest server, which is in Ireland. Ivan browses for cars and then chooses the color, seat fabric, and window tinting, all from the server in Ireland. After perfecting his future vehicle, Ivan saves his customization and offers a price to start the negotiation process.

Ivan's car customizations and initial bid transmit to the interpreter that is running the BUI application in Ireland. The BUI application's write connection in Ireland then redirects Ivan's data to the primary database in Virginia. As the database in Virginia updates, the updated records replicate back to the local copy of the database on the machine in Ireland. Because the process only takes a couple of minutes, Ivan will be able to review his order with all the details displayed directly from Ireland's local database.

The More BUIs the Merrier

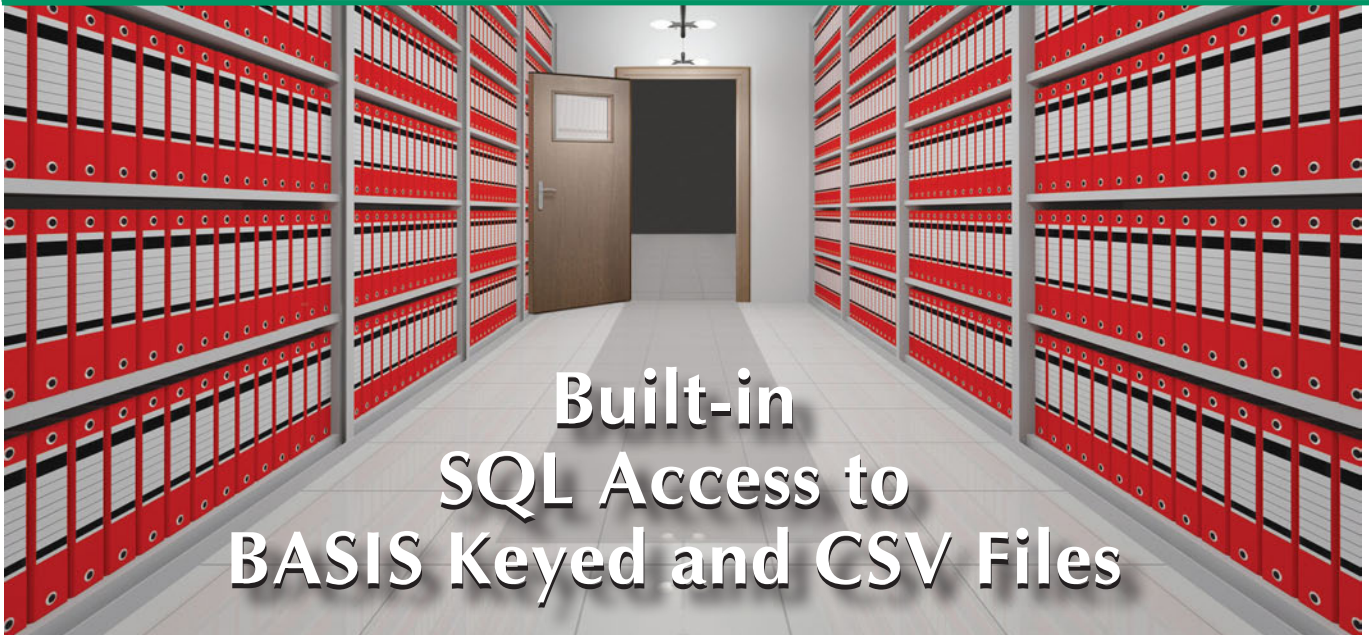
Amazon's geo-aware DNS addresses and BASIS' replication feature come together to allow BUI applications to perform quickly, no matter where in the world the user runs them. The best part is that with replication, most of the application maintenance occurs on the source and propagates to the target machines.

After Henry setup the other servers and the replication jobs, he still only has to maintain and update a single server. All the other servers are maintained automatically. So, while this adds some time to the initial deployment, after it's up and running Henry doesn't have to do any more work to get updates to all the servers than he does to a single server.

The result is a distribution of a BUI application that is completely transparent to customers around the world. For all they know, it is running on a single server in their own backyard! ■



Read Are You Prepared for Cloud Failure? at links.basis.com/12cloud



Built-in SQL Access to BASIS Keyed and CSV Files

Built-in stored procedures now provide SQL access to BASIS data files or CSV files with the familiar SQL 'CALL' syntax and forgo the typical prerequisite of a BASIS data dictionary or the need to write SQL syntax. Any SQL-enabled application can now access BASIS data files using this new functionality. Built-in SPROC's are an integral part of the BASIS RDBMS; the built-in BBJSYS database provides access to the new built-in SPROC's. The SQL 'CALL' syntax accesses built-in SPROC's through the BBJSYS or any other database.

Introducing the new SPROC's

These new built-in SPROC's, available in BBj® 12 and above, are always available and called from a connection to any available BBj database. If there is no database defined within BBj Services, use the system database to execute the stored procedure. The 'BBJSYS' database (built-in system database supplied with BBj Services) will execute the stored procedure without any need for a data dictionary.

BBj or third party applications can use the BBj ODBC Driver® or JDBC Driver to access the built-in SPROC's via the BBJSYS database. Here is a sample JDBC connection URL to the BBJSYS database:

```
jdbc:basis:localhost?DATABASE=BBJSYS&SSL=false&USER=admin&PASSWORD=admin123
```

Understanding how new SPROC's Work

Built-in stored procedures make use of a user-supplied file location and record template as parameters to the SQL CALL statement to return a result set based on the data from the file. The SQL syntax for use with the built-in stored procedure looks like this:

```
GET_RESULT_SET (file_path, template)
```

"File_path" is the path to the data file you wish to reference and "template" is the string template that describes the structure of the file.

Using the new SPROC's

The following simple example retrieves information from the ChileCompany's CUSTOMER file (an MKEYED file) located in the <BBjHome>\demos\chiledd\data directory:

```
CALL GET_RESULT_SET
('C:/Program Files/basis/demos/chiledd/data/customer', 'CUST_NUM:C(6), FIRST_NAME:C(20), LAST NAME:C(30),
COMPANY:C(30), BILL_ADDR1:C(30), BILL_ADDR2:C(30), CITY:C(20), STATE:C(2), COUNTRY:C(20), POST_CODE:C(12)')
```



By Robert Del Prete
Quality Assurance
Engineer

Executing this SQL statement in Enterprise Manager returns the data from the CUSTOMER file as an SQL result set, as shown in **Figure 1**.

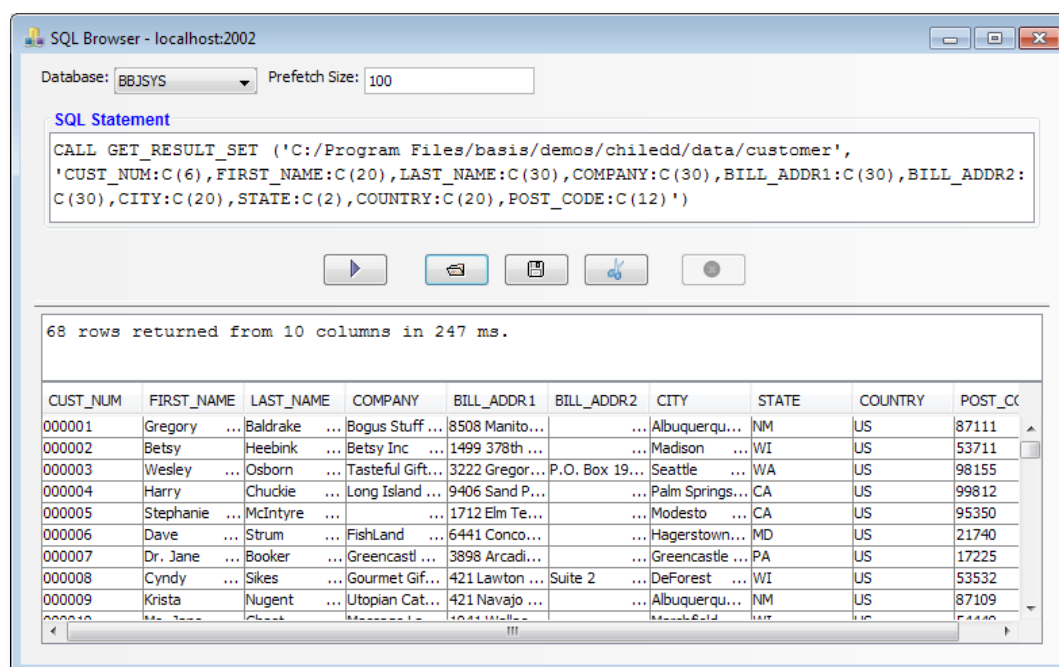


Figure 1. The built-in SPROC via SQL in Enterprise Manager

The call in **Figure 1** to the GET_RESULT_SET SPROC returns the first ten fields, as defined by the provided string template, from all records within the CUSTOMER file. This is extremely helpful when a data dictionary is not available for the data file. Utilizing iReport together with the built-in GET_RESULT_SET SPROC allows developers to provide stunning reports without the overhead of creating a data dictionary or writing SQL syntax for their customers' traditional file system-based data.

Understanding Advanced Data Manipulation

BASIS' SPROCs are very versatile and developers can utilize the SQL language to treat a SPROC's result set as a table to join, group, filter, and order the returned data. SQL clauses such as WHERE, ORDER BY and JOIN allow programmers to maximize their control over the returned data to suit the needs of the application and end users.

The example in **Figure 2** uses the GET_RESULT_SET stored procedure to pull data from two different files. The statement joins the results of two CALLs to GET_RESULT_SET SPROC, utilizing the WHERE clause to match the account numbers in the two tables. It then uses a GROUP BY to group the returned data by ACCOUNT_NUM and DESC, finally ordering the result set by account number.

```
SELECT
  GLMAST.ACCOUNT_NUM, GLMAST.DESC, COUNT(GLMAST.ACCOUNT_NUM) AS "ENTRIES",
  STR(SUM(GLJOURN_DET.AMOUNT), '$#,###,###.00') AS "Balance"
FROM
  (CALL GET_RESULT_SET
    ('c:/Program Files/basis/demos/chiledd/data/GLMAST',
    'ACCOUNT_NUM:C(6),DESC:C(35):')) as GLMAST,
  (CALL GET_RESULT_SET
    ('c:/Program Files/basis/demos/chiledd/data/GLJOURN_DET',
    'JOURNAL_NUM:C(6),LINE_NUM:C(4),ACCOUNT_NUM:C(6),MEMO:C(30),AMOUNT:N(10)')
  ) AS GLJOURN_DET
WHERE
  GLMAST.ACCOUNT_NUM = GLJOURN_DET.ACCOUNT_NUM
GROUP BY
  GLMAST.ACCOUNT_NUM, GLMAST.DESC
ORDER BY
  GLMAST.ACCOUNT_NUM
```

Figure 2. Advanced SQL query utilizing the built-in SPROC to pull data from two files

This SQL statement returns the account numbers in ascending order along with the description for each account, the number of entries, and the current balance on the account without the use of a data dictionary, as shown in **Figure 3**.

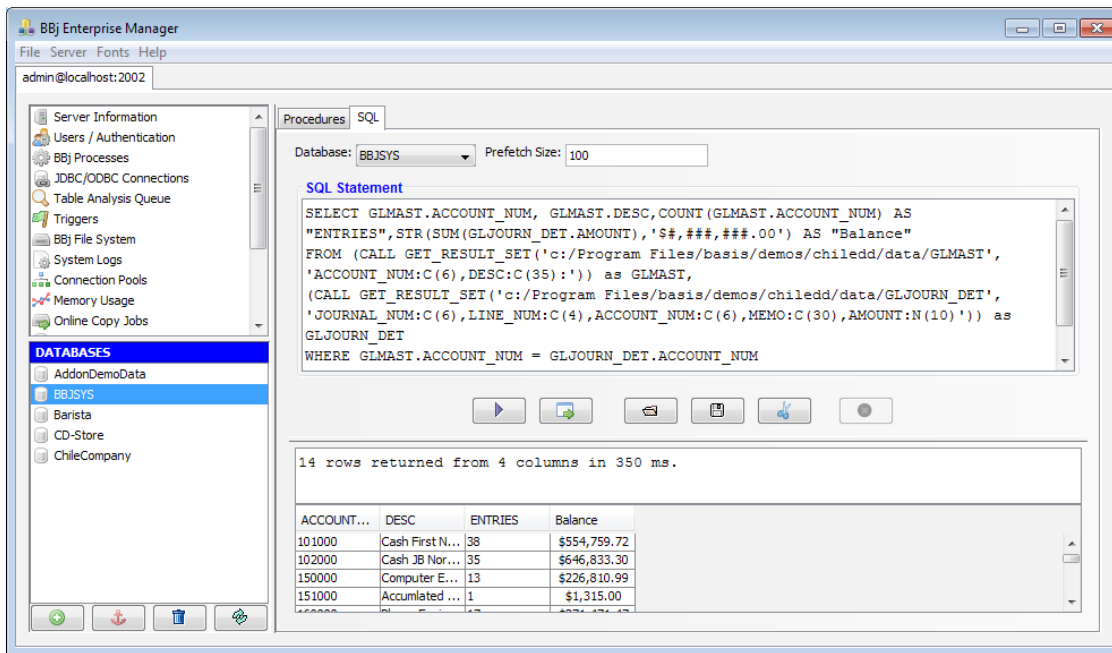


Figure 3. SQL using the built-in SPROC in Enterprise Manager

Retrieving Data From a CSV File

A second type of built-in SPROC is also available for returning data from a standard CSV (comma-separated values) file using this syntax:

```
GET_RESULT_SET_CSV (file_path, template, delimiter)
```

As with the previous built-in SPROC, the `file_path` parameter denotes a fully qualified path to the CSV file. The template used for a CSV file simply describes the fields for each row and the delimiter determines the separator between each row of data in the text file. The SPROC expects the actual bytes of the delimiter rather than a string representation of the characters. In the case of a linefeed, a `CHAR(10)` would be specified in the SQL statement instead of the common `'\n'` string representation of the newline character. The same sorting and filtering capabilities are available to this SPROC as well. Consider the following CSV file format:

```
Baroque,George,Fredric,Handel,1685,1759
Classical,Joseph,,Haydn,1732,1809
Romantic,Carl,Maria,von Weber,1786,1826
```

Creating the SQL statement to access data in the CSV text file via the SPROC is easy. Insert the file path and create the string template by following the format of the CSV file. Include the delimiter for the records, in this case a line feed represented by `CHAR(10)`. Filter the composers using `MUSIC_TYPE` and exclude any born after 1800. Lastly, group the results according to the composer's last name. The SQL statement code in **Figure 4** uses the built-in SPROC for a CSV file, selects all of the fields, sorts by music type and birth date, and then orders by the last name.

```
SELECT
MUSIC_TYPE, FIRST_NAME, MIDDLE_NAME,
LAST_NAME, BIRTH_DATE, DEATH_DATE
FROM
(CALL GET_RESULT_SET_CSV
('c:\users\data\desktop\composers.csv',
'MUSIC_TYPE:C(10*), FIRST_NAME:C(10*), MIDDLE_NAME:C(10*), '+
'LAST_NAME:C(10*), BIRTH_DATE:I(4), DEATH_DATE:I(4)',
CHAR(10))
) AS COMPOSERS
WHERE
MUSIC_TYPE='Classical' AND BIRTH_DATE < '1800'
GROUP BY
LAST_NAME
```

Figure 4. Sample of SQL using the built-in SPROC for a CSV file

The result of the code sample appears in **Figure 5**.

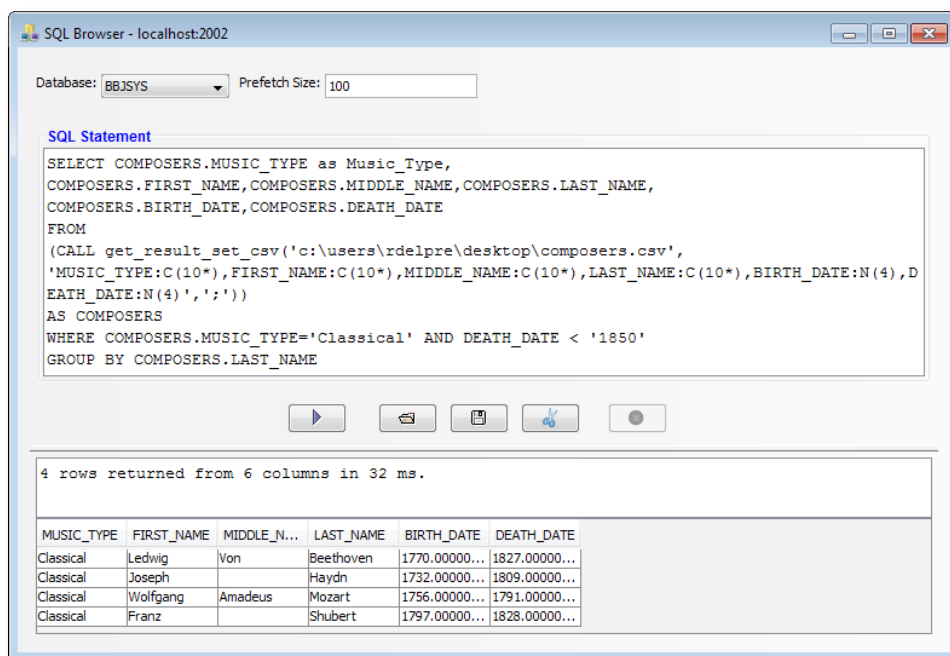


Figure 5. SQL using the built-in GET_RESULT_SET_CSV SPROC

Summary

The new built-in SPROCs allow access to any BASIS or CSV formatted data files regardless of the presence of a data dictionary. Developers can leverage this new feature in many ways, supplying new access to data by end users. Being able to filter and sort the data using SQL syntax also provides additional flexibility for the distribution and presentation of that data. ■



- Read more about built-in SPROCs in the online documentation at links.basis.com/bisprocs
- Check out these articles
 - *Using Stored Procedures to Add Business Logic to the Database* at links.basis.com/06sprocs
 - *Unleashing the Power of SPROCs Without SQL* at links.basis.com/07sprocs
 - *Recipes for Successful Report Writing* at links.basis.com/09reportwriting

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



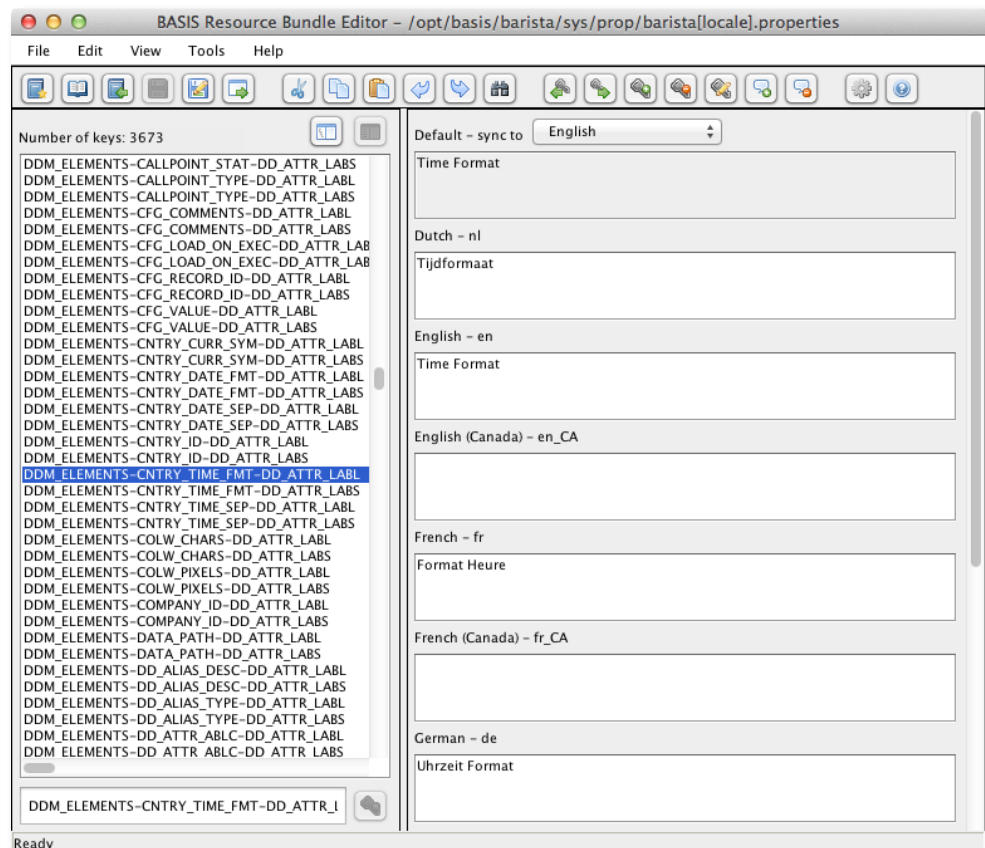
Babbling With the New Bundle of Joy

BASIS' New Resource Bundle Editor

Many believe that removing the string literals displayed to users from application code is only necessary in applications that will be used in multiple languages. Although this is the common case, there is a major reason to remove the literals even when writing an application for a single language. Maintenance of these string literals becomes very easy when they are separated from the code; so easy in fact, that non-programmers can even change the text. Another benefit to separating strings from the code is that a single update of the string propagates the change throughout the entire application regardless of how many times it appears. If the need ever arises to internationalize the application, simply send the resource bundle off to a translator. In BBj®, a developer can separate the text literals from the code by using the BASIS resource bundle utility called the Resource Bundle Editor (RBE) that facilitates easy creation and maintenance of resource bundles. See **Figure 1**.

Overview

A resource bundle is simply a collection of one or more “property files” or ASCII files that contain key/value pairs much like a string table in BBj. The names of property files include the name of the resource bundle and, if necessary, locale information. For example, a HelloWorld application might have a HelloWorld.properties file that contains default values in English and a HelloWorld_de_DE.properties file for German. The RBE is a BBj application building block utility program that provides a very intuitive user interface and is a turnkey solution for creating, editing, and updating resource bundles and their associated translation property files.



By Brian Hipple
Quality Assurance
Supervisor

Figure 1. The Resource Bundle Editor displaying a few translations of a particular key

Conceiving

To create or edit a resource bundle, use the menu items or toolbar buttons inside the RBE. A most recently used list is also available from the menu to quickly reopen resource bundles. When creating a new resource bundle, a dialog box like the one shown in **Figure 2** prompts the user for the folder and base name of the resource bundle. A locale is the language/country/variant and is selectable from the list or entered manually. The locale en_US is US English rather than en_GB (Great Britain) English for United Kingdom or en_CA English for Canada.

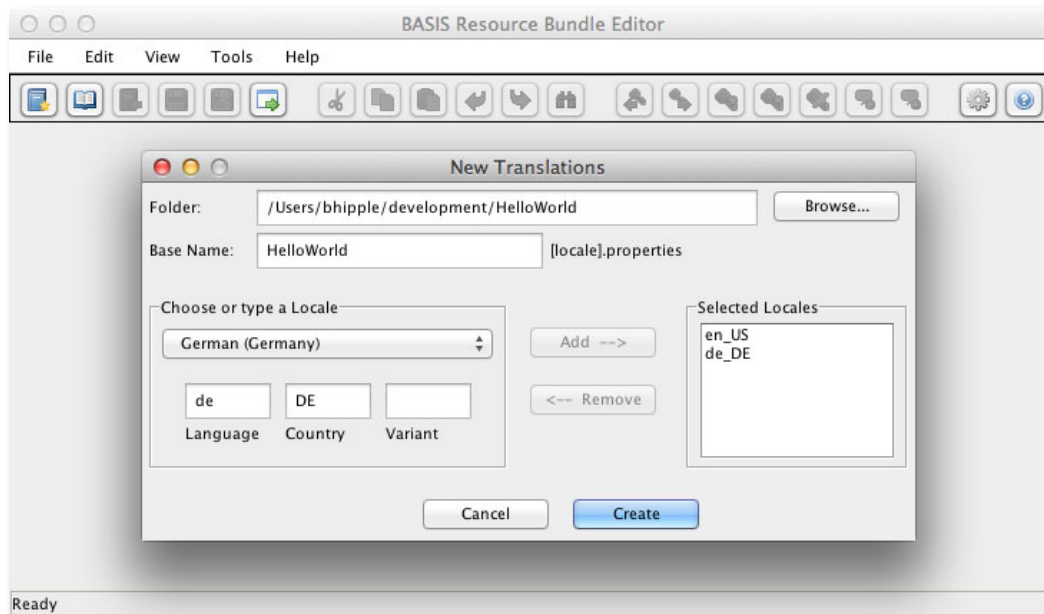


Figure 2. Define a new resource bundle

Reproducing

After creating or opening a resource bundle, users can easily add keys in several ways – the [Add Key] button, toolbar button, menu item; or by right mouse clicking and selecting the Add Menu option from the popup menu. A dialog appears in which the user can enter the new key name as shown in **Figure 3**.

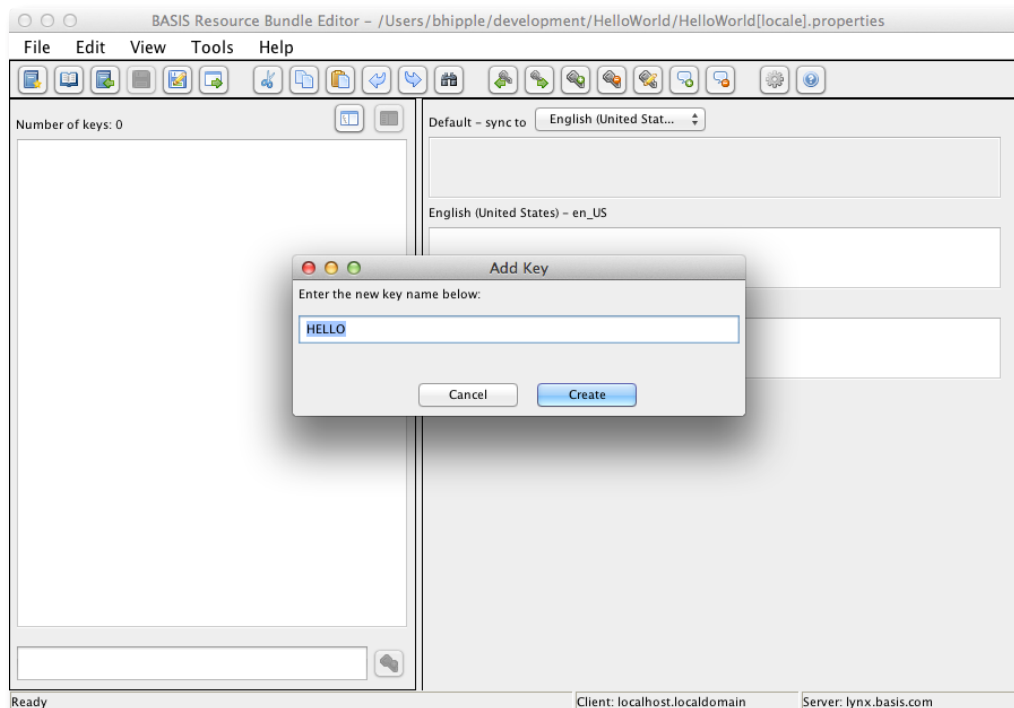


Figure 3. Add a new key to the bundle

Localizing

Next, specify the values for the default and any additional locales. The utility provides the option to sync to a default locale as shown in **Figure 4**.

Extra Capabilities

The keys display one of two views; in a straight list or, if there is a hierarchy in the key name, in a tree view. The utility offers a myriad of tool buttons and menu items to open/close/save resource bundles, copy/cut/paste values, undo and redo values, traverse keys, add/remove locales and keys, and set options.

An advanced “Find” dialog (see **Figure 5**) helps to find keys/values, which is a useful feature when working with very large resource bundles.

Propagating

After creating the resource bundle, the developer can access it from inside a BBJ application in one of two ways; using the BASIS-provided [BBTranslator](#) utility or the Java ResourceBundle API. The BBJabber utility facilitates the creation of resource bundles from BBX program source and resources. This includes adding the necessary code to access the resource bundle via the BBTranslator utility instead of using string literals. Take a look at some additional *BASIS Advantage* articles – [Can Your App Speak to Your Customer?](#) and [Parlez-BUI Français?](#) – to learn more about this BASIS-supplied utility.

BBTranslator Utility

The BBTranslator utility offers several advantages. One benefit is that developers can use BBJ code without learning another language. The BBTranslator includes object-

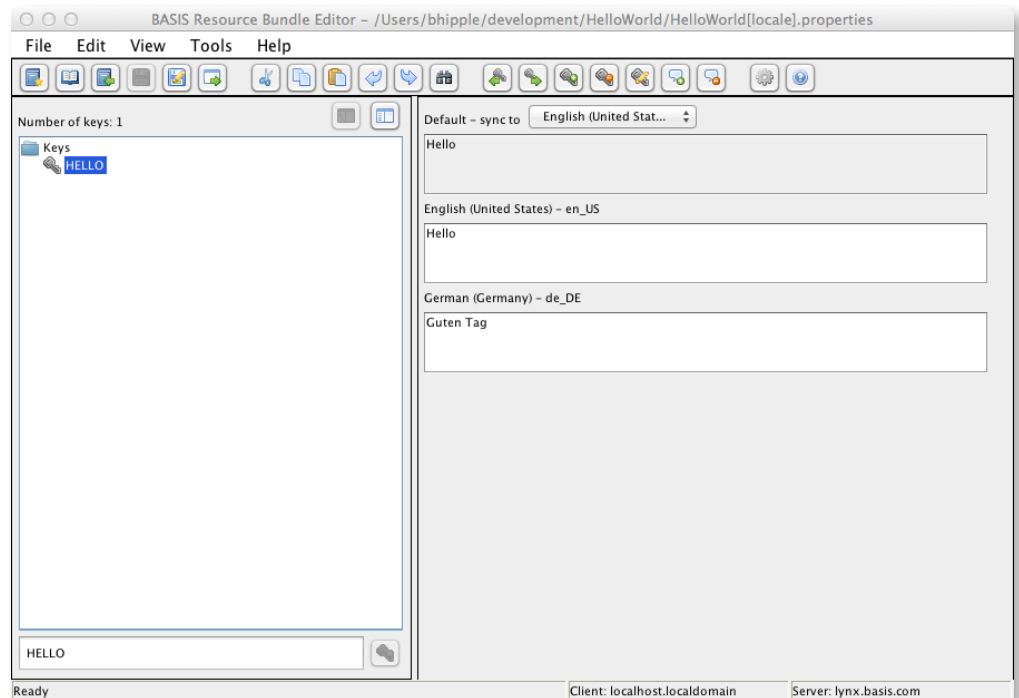


Figure 4. Set values for the new key

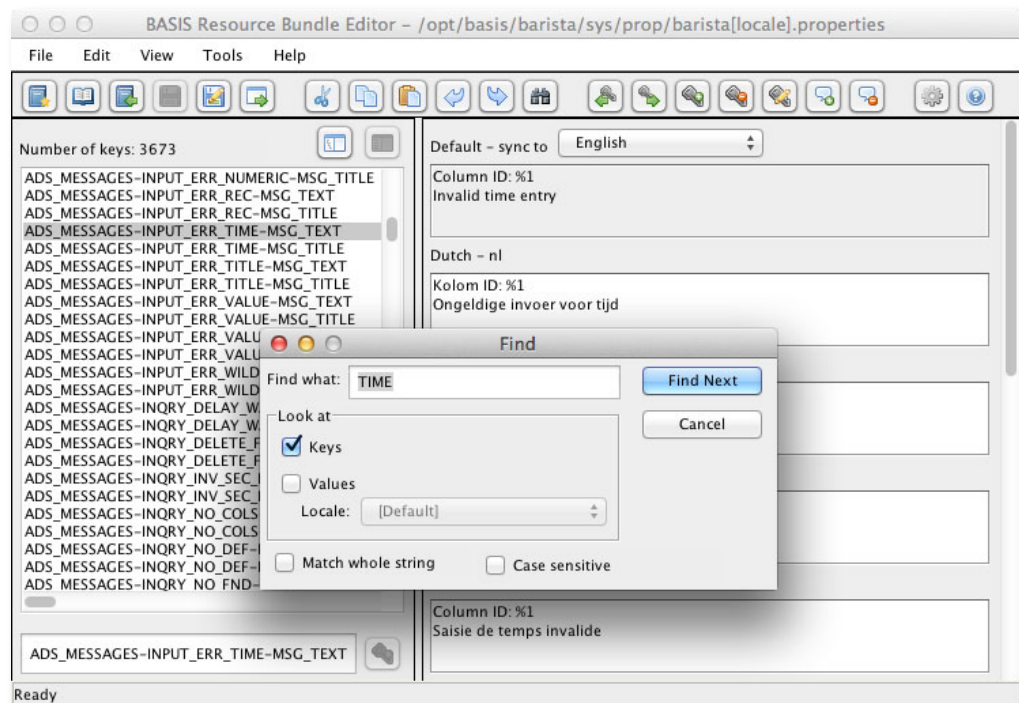


Figure 5. Find a key or value in a bundle

oriented methods (**Figure 6**) and CALL'able subroutines (**Figure 7**) to make the access to the resources bundles from legacy or object-oriented code as seamless as possible. Another significant advantage is that the resource bundle may exist in a directory referenced in the PREFIX.

```
use ::bbtranslator.bbj::BBTranslationBundle
use java.util.Locale

bundle! = BBTranslationBundle.getBundle("HelloWorld")
resourceBundle! = bundle!.getTranslations(new Locale("de","DE"))
resourceBundleValue$ = resourceBundle!.getTranslation("HELLO")
print "Resource bundle value: " + resourceBundleValue$
```

Figure 6. Use the BBTranslator's object-oriented methods to retrieve a translation for a key

```
call "bbtranslator.bbj::getInstance",resourceBundle!,"HelloWorld","de_DE","", ""
resourceBundleValue$ = resourceBundle!.getTranslation("HELLO")
print "Resource bundle value: " + resourceBundleValue$
```

Figure 7. Use the BBTranslator's CALL'able subroutines to retrieve a translation for a key

Developers who prefer programming in Java can use the [Java ResourceBundle API](#) to access the resource bundle. However, this method requires that the bundle is in a jar referenced in the BBJ Classpath using a [session-specific classpath](#). Developers can localize JasperReports using resource bundles packaged and referred to in this manner. Although Java code is necessary for this option, it is straightforward and almost as easy to implement as the CALL approach. ☺ See **Figure 8**.

```
use java.util.ResourceBundle
use java.util.Locale

resourceBundle! = ResourceBundle.getBundle("HelloWorld", new Locale("de","DE"))
resourceBundleValue$ = resourceBundle!.getString("HELLO")
print "Resource bundle value: " + resourceBundleValue$
```

Figure 8. Use the Java ResourceBundle API to retrieve a translation for a key

Summary

Whether or not you have internationalized your application, it is always a good idea to keep displayed string literals separate from the application code. The RBE is a great way to manage the resource bundles that contain this text. Since the RBE is a BBJ utility, it can run in Java Web Start or in a browser (BUI mode) to facilitate direct exchange from professional translators to the resource bundle.

Before RBE, translators relied on a tool like MS Excel, which created an extra step for the developer to manually put the text into the application. Alternatively, translators could have used a resource bundle-aware IDE but most do not provide a consistent manner for handling the resource bundles. Another drawback to the second approach is that installing an IDE is a very heavy-handed requirement for non-programmers to perform translations.

Now, all one needs to do is to send the translator a link to the RBE application! Honestly, how can that be any easier? BASIS uses the RBE to manage resource bundles for the BASIS Custom Installer, the BASIS Product Suite Download page, the Barista RAD tool, and for the AddonSoftware building block. Won't you welcome this new bundle and add it to your family of tools? ■



- For more information
 - BBTranslator in the online documentation at links.basis.com/bbtranslator
- Read about
 - BBJabber in *Can Your App Speak to Your Customer?* at links.basis.com/09appttranslate
 - BUI localization in *Parlez-BUI Français?* at links.basis.com/11builocal



An Insider Look at BASIS Testing

The biggest bane of software development is releasing a product before it has been properly tested or containing an array of bugs caught only after the release. At BASIS, our number one priority is producing the best product that we possibly can. Although we use many tools to achieve this goal, the most crucial is our test suite that we designed specifically to test our code changes at all levels of production to ensure that our products are released with the highest quality. We run the suite automatically after a new build compiles, which begins within 8 minutes of any checkin to the SVN source code archive. Our current test suite has two parts - **JUnit** and testbed.

BASIS uses JUnit as our first level of defense to ensure that any new code development does not alter or affect other parts of the code currently working and operating within BBj® Services. JUnit, originally written by Erich Gamma and Kent Beck, provides an open source tool that comes standard with Java and is a simple framework for writing repeatable tests. JUnit is an instance of the xUnit architecture for testing frameworks and contains three distinct parts - assertions for testing expected results, test fixtures for sharing common test data, and test runners for running

tests. By running the repeatable tests before we check in any code, JUnit proves that the new code integrated with the existing code will not produce any unexpected errors.

The second level of defense, better known as the testbed, contains the test server and client that we designed to establish that the actual product is fit for general release. The test server is responsible for running BBj Services, displaying the GUI, and executing the BBj test code. The test client's sole responsibility is to log and record the information obtained while the test server runs the code tests. This level of testing is similar to the way BASIS developers test their own BBj applications.

So how do we actually use these tools? Well, the first step is to compile BBj Services so that it runs properly on all the platforms that we support. Specifically, we take the code that we checked in during our "first level" tests, compile it on the oldest supported platform version, and then run BBj Services on the newest supported platform. For example, Microsoft's oldest supported OS is Windows XP while its newest released OS is Windows 8. Therefore, we compile on Windows XP and run BBj Services on Windows 8. The same would apply to Linux, UNIX, Solaris, and so on. This practice ensures that BBj Services will not require the latest OS specific features and tags that would limit backward compatibility. As a result, BBj Services can run on all platforms supported by the operating system vendor.

The second step in our testing process is to write the BBj code that will execute the new features. If you are thinking we just write a simple program to test the new additions to our product, you are half right. The first BBj program usually tests the new feature itself using our product documentation as a guide. The next several tests integrate the new feature within our old tests to increase the level of complexity. In fact, many of the BBj test programs appear as code samples in the online BASIS documentation.

Finally, we run BBj Services on the newest supported platforms with our test BBj code. If the tests have a user interface, the UI will appear on the test server with the test client logging the results of the tests. We then analyze the logs and review for corrections. After making any necessary changes, we rerun the tests and repeat the process until it is error-free.

Why go through of all this effort? This meticulous testing allows us to see the actual code and controls in action on the screen as if we were running it in a real-world production environment. As a company in the modern marketplace, BASIS is committed to producing the most modern, up-to-date, cutting edge software, and strives to include modern programming features. We take great pride in what we do and the test suite ensures that we can accomplish our goals while continuing to meet yours. ■



By Aaron Wantuck
Software Engineer



For more information on

- JUnit - refer to junit.org or download it from bit.ly/7ESsE1
- Writing JUnit tests - visit bit.ly/SSaSh2



'Git'dy Up, Developers!

If you're not familiar with free, open source versioning tools, you might think we grossly overlooked a misspelling in the title and used poor grammar to boot! Now that we have your attention, read on.

Recently, BASIS incorporated Git distributed revision control and source code management (git-scm.com) into the AddonSoftware® by Barista® (Addon) upgrade procedure to make upgrading customized Addon packages easier than ever before. In this article, we examine the Addon upgrade process prior to the introduction of Git,

what Git is, how it adds new intelligence to the Addon upgrade process, and Git's overall role in an Addon upgrade.

Upgrading Customized AddonSoftware Installations

One of the advantages of developing Addon within the Barista Application Framework is Barista's application project structure. Developers wanting to make customizations to Addon can create a new Barista project and save customized forms, callpoints, reports, publics, custom classes, etc., inside that project file structure. This separation ensures that it preserves customizations when upgrading Addon. After installing a new version of Addon, the Barista Install Application Wizard (IAW) allows developers to re-install these custom projects by importing customized forms and/or data tables back into the standard product.

While this is the recommended process for making customizations, it is possible that some developers have modified the standard product directly or have blended some modifications in a separate project file structure with others made directly to the core product. In addition, the emphasis of the Barista IAW is on forms and/or data tables. The wizard doesn't reconcile changes in callpoint code or other code not related to the user interface.

Regardless of how customizations were made, the result is that Addon's modifications suit customers' needs. In many environments, such customizations leave the product "stuck in time," with no possibility of an application upgrade. With Barista and Git, developers can now modify both the user interface as well as other code, and still realize the benefits of an upgrade!

Life Before Git

Prior to Git, developers could use the Barista IAW to incorporate form or table modifications back into standard Addon. However, there was no automated way to preserve those customizations if they were made directly against the Addon installation (i.e., in the Addon file structure itself). In addition, the wizard is not concerned with changes in other source code files such as callpoints, reports, publics, etc., so there was no easy way to see changes in the standard code and determine if they should be included in the customized counterparts. In terms of callpoints, additional code that developers may have added to run before or after the standard



By Shaun Haney
Quality Assurance
Engineer



By Chris Hawkins
Software Developer

callpoint may be fine, but if the developer took a copy from a standard callpoint and augmented it to run instead of the standard, there was no easy way to see if subsequent changes to the standard should be incorporated into their “instead-of” callpoints. You can imagine that analyzing source code could be a tedious process; VARs either had to perform three-way comparisons by hand – comparing the original Addon to the customized version and then to the new Addon – or write their own in-house scripts to upgrade their code.

In the pre-Git life, there was no single, comprehensive process for upgrading Addon. Each process was specific to the company requiring the upgrade. Furthermore, manual upgrades delivered incomplete or “hybrid” versions of Addon where not all of the customer’s Addon source is upgraded, resulting in files from different versions of Addon coexisting in the same installation. Last but not least, when using in-house procedures for upgrades, there often was no mechanism for remembering or recording decisions made in past upgrades. In these cases, the same questions, like whether to replace a piece of code with a newer version, typically resurfaced repeatedly with each upgrade.

What is Git?

To address the dilemma of having highly customized code in an old version of Addon, BASIS incorporated Git into Addon’s upgrade process. But how exactly would Git solve this?

Git is a distributed source management control system. [Linus Torvalds](#) and other open source developers originally designed Git as a replacement for [BitKeeper](#), previously the versioning system of choice for maintaining the Linux kernel. Much like [CVS](#) and [Subversion](#), Git tracks file changes so users can access past versions of their files. Unlike Subversion and CVS, Git is a distributed revision system in which the user actually obtains a copy of the entire archive and works with their copy locally, rather than only checking out a single revision of the files from a central server. Having the entire archive instead of a single revision actually allows the user to keep their own custom version of the archive, while still allowing that user to get updates from the original archive or even other customized copies of the original archive.

So, how exactly does this work? Let’s first look at a generic non-Addon scenario for using Git. Let’s say your favorite free text editor is available from a public Git archive and you decide that your company needs some specific enhancements for highlighting part numbers with a color keyed to the department that produces that item. Since this feature is unique to your company only and the editor does not provide plug-in support, you decide to “clone” the archive and add the needed functionality into your own version of the editor. So you use the `git clone` command to copy the Git archive to your system. Then you checkout the latest version of the source code that actually appears in your Git directory so you can make changes in place. You add the features you want and then check them in. A new checkin appears in your archive that is not present in the remote archive.

A few months pass, and you read that they’ve enhanced the way that the text editor finds and highlights text. You want this enhancement but are concerned that some of their changes will overlap yours. This time, you pull the new version from the remote archive and Git attempts to merge its changes on top of yours automatically. Wherever you have changed code but the remote archive hasn’t, your change is preserved. Wherever the remote archive has changed code, but yours has not, the remote archive change is automatically applied. However, if there are changes on the same line in both archives, Git denotes the conflict and will not complete the merge as a brand new revision until you resolve the conflict and commit your change. The total number of conflicts is typically only a small fraction of the total number of merges Git performs.

Git’s automatic merge process allows you to focus only where it actually needs your intervention – on pieces of code that you changed and have also changed in the original source. Once you resolve the conflicts and commit the files, Git creates a new revision of the code that incorporates both your customizations and the original source. In this way, Git allows you to maintain custom code while still being able to incorporate upgraded code from the original source.

Git Brings New Intelligence to the Upgrade Process

With its ability to perform automatic merges, detect conflicts, and remember custom changes, Git speeds up the Addon upgrade process and ensures a complete upgrade. The basic steps in the Git cycle are listed below and illustrated in **Figure 1**, in which Addon revisions in Git appear as layers of an onion. Specific processing steps depend on whether customizations were made directly into the Addon source code or saved in a separate Barista project file system.

1. Clone the Git archive directly from BASIS. The Git archive contains every version of Addon.
2. Roll the archive back to the same version as the customer’s current version of Addon.
3. Copy the customized Addon code into the archive.
 - If developers made customer customizations directly to the Addon file structure, then they could copy that core code into the archive as a new revision.
 - If developers maintain the code in a separate Barista project, BASIS has created a set of utilities that copy the code modifications into the archive. Developers can then check these modifications in to recreate the new revision.
4. After checking in the new revision, the developer rolls the archive forward to incorporate any changes made since the customer’s current version all the way to the new version. Most of the changes will merge in smoothly, but a handful of changes will likely result in conflicts.
5. Review and correct the conflicts, and commit each resolved file. Once the files are committed, Git has a brand new revision containing the customized code and upgraded code that comes with the latest version of Addon.
6. Move the upgraded code back from the Addon archive to the customer’s copy of Addon.
 - If developers made customer customizations directly to the Addon file structure, they can copy the upgraded code directly to the Addon core.
 - If developers maintain customer code in a separate Barista project, BASIS created a set of utilities to move the upgraded code back to the Barista project.



Figure 1. The basic steps in the Git cycle

At this point, keep and use the Git archive for the next Addon upgrade so that conflicts that occurred this time around will not need revisiting in the future.

‘Git’ the Big Picture

While Git plays a critical role in Addon’s upgrade process, it is just one step in the whole process. Now that you’re familiar with Git, let’s review the entire process.

Download and Install AddonSoftware by Barista

The initial installation places Addon in the same directory structure as BBj®. BASIS intended this copy for demo or evaluation purposes and does not recommend it for a production installation. Once installed, use the AddonSoftware Install/Upgrade Wizard (AIUW) to make a new instance of Addon outside of the BBj home directory. This is the live/production version of Addon.

As mentioned, the recommended process for customizations is to use the Barista Create Application task to set up a separate Barista project structure for your modifications. You will save or create some files directly in the new project, and others will be saved there automatically by Barista when using application replication mode from the Barista Form Manager (see [Customizing Barista Applications](#) for more).

Upgrade AddonSoftware

When you’re ready to upgrade to a new version of Addon, download and install the new Addon into BBj home (overlying the current demo or eval copy). Then use the AIUW to perform the upgrade process. Note that the AIUW facilitates parallel operations so users can continue to run the live production copy of Addon during the upgrade process. Read more about the AIUW in [AddonSoftware Installation and Upgrade Processes](#), but in short, this wizard chains together the following tasks:

- Make a new copy of Barista and Addon in a user-specified location
- Copy backed-up administrative and syn file data from the live production installation
- Run the Auto-synchronize process
- Copy and/or install other Barista projects (customization projects or verticals)

Copy, Upgrade, Install the Project

In order to upgrade the customization project via Git, opt to copy, not install, that project. Once the project is copied, you’ll use the Git process to upgrade the project, and then use the Barista IAW to install it into the new Addon.

If you are not using a separate Barista project for customizations, then after using the AIUW to create your new instance of Addon, perform the Git processes and then copy the desired files directly into the new copy of Addon.

Conclusion

While customer installations vary and Git may not be the silver bullet that slays all villains, the incorporation of Git into the Addon upgrade process is a huge step towards standardizing Addon’s upgrade process and greatly simplifying the arduous task of finding and incorporating Addon’s upgraded code into a custom installation. Git also makes future upgrades even easier than the initial upgrade by ensuring a complete upgrade and remembering the decisions made in previous upgrades. For those developers who have put off an Addon upgrade due to the difficulties involved, Git is race horse that will “get” you across the finish line.

Hop on today and git-dy up! ■



For more information, read

- *Create Vertical and Customize Applications, Parts 1, 2, 3*
links.basis.com/baristaref
- *AddonSoftware Installation and Upgrade Processes*
links.basis.com/addoninstallupgr



Platform-Independent Task Scheduler

There are numerous reasons why an administrator or developer might want to schedule particular tasks to run automatically at a particular point in time or at a regular interval such as a nightly backup job, maintenance utility, or some kind of batch processing job. One common method for scheduling tasks is the UNIX or Linux cron job. However, certain types of tasks (especially those which require interaction with BBJ® Services) would be more easily managed if BBJ had a built-in scheduling feature and it would also remove operating system dependencies from an administrator's deployment plan. Therefore, BBJ 12 introduced a new scheduling feature to the already robust set of tools available in the BBJ Enterprise Manager as well as the Admin API.

Two components make up the scheduling system: Task Groups and Tasks. A Task Group contains a list of one or more tasks to be executed synchronously in sequential order, when it should first

execute, and how often the Task Group should repeat execution. A Task is a single item to be managed and executed by a Task Group. The best way to understand the scheduler is to simply walk through an example.

Scheduling Example

The example in this article illustrates what might occur when making a weekly backup of data files located on a replication target for zero downtime on the live system. Outside the BBJ scheduler, this particular job would require additional Java or BBJ coding to handle pausing and resuming the replication job but since the scheduler is designed with BBJ in mind, it requires only a couple of mouse clicks to perform these tasks.

The Scheduling panel in the Enterprise Manager displays a list of all currently configured Task Groups and provides an interface for creating, modifying, and removing them. The panel shows the tasks within each group, when they will run next, and when last run. Access the Scheduling panel by selecting the "Scheduling" item from the Enterprise Manager navigation area.

Figure 1 shows four Tasks that make up this Task Group. The first task pauses the replication job, which is not as easily done from something like a cron job. This ensures that the replication target is in a clean state and ready for backup. The second task executes a BBJ program that could do anything needed to prepare the data for backup. Next, the scheduler executes a system call; in this case executes a Windows .bat batch file to perform the backup operation. Finally, the last task automatically resumes the replication job so that the target can catch back up with the source.

Task Group	Scheduling / Task Type	Next Run	Last Run	Associated File
MyJob	5/13/12 1:44 PM weekly	5/13/12 1:44 PM		
●	Pause Replication Job			ChileRep
●	BBJ Program			C:\mypath\myprogram.bbj
●	SCALL / Script			backupsomefiles.bat
●	Resume Replication Job			ChileRep

Figure 2 shows the configuration of the task group which executes weekly, every Monday and Friday at 1:44 pm. Note that specifying an "End" date would cause the Task Group to cease executing at that particular point in time and a change to the "Repeat Every" field allows for skipping weeks.

Four types of Tasks are available with the scheduler: Run BBJ Program, Execute System Call, Pause Replication, and Resume Replication. Each Task type has a different set of configuration settings as shown in **Figure 3**.



By Jeff Ash
Software
Engineer



By Nick Decker
Engineering
Supervisor

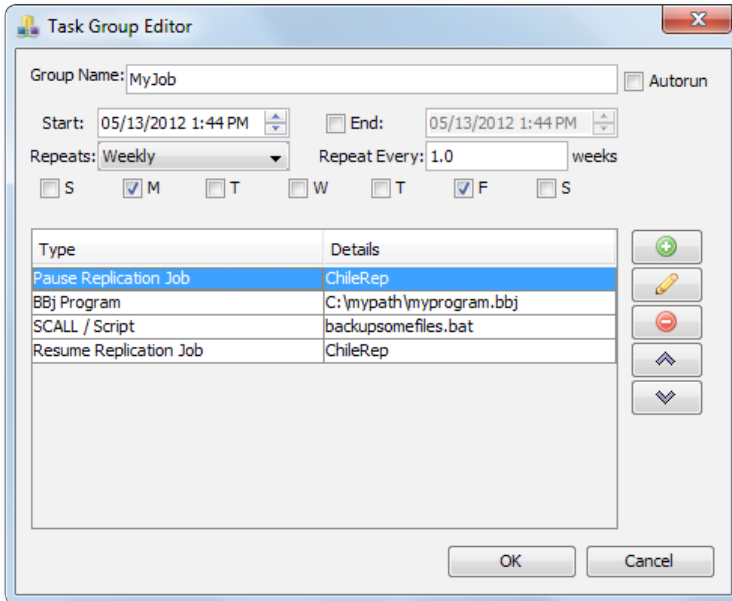


Figure 2. Task Group Editor

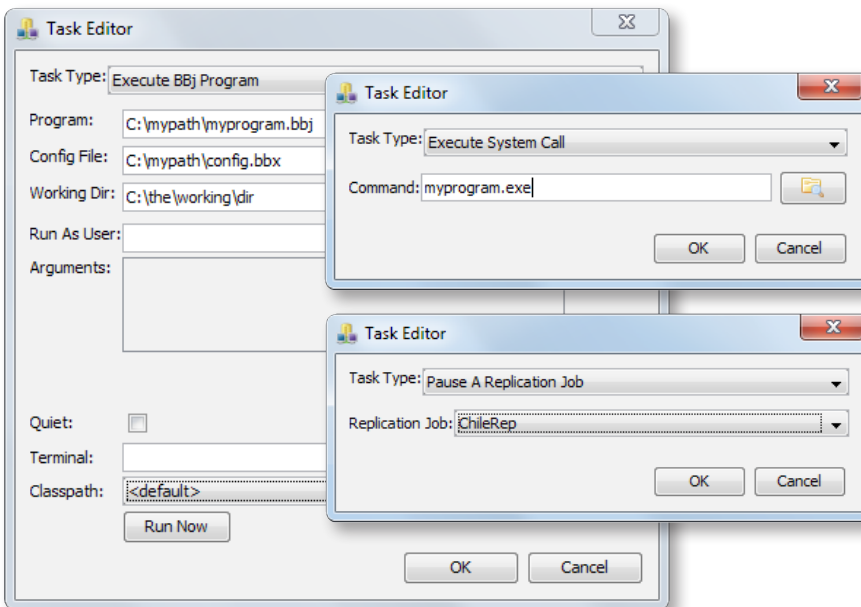


Figure 3. Task Editor showing Run BBj Program (left), Execute System Call (top right), and Pause A Replication Job (bottom right)

Another Use Case

When we first designed the new BUI-based BASIS Product Suite Download page (links.basis.com/getbbj), one of our goals was to integrate it seamlessly with the automatic build system in the cloud. The BUI program accomplishes this by utilizing a custom BBj class that dynamically retrieves the available released and development builds from an Amazon S3 bucket (S3 is the nickname for Amazon's Simple Storage Service, which we think of as a large hard drive in the cloud). The end result is that the download page lists all available BBj downloads automatically, without any human intervention. The code worked perfectly, but subsequent performance analysis revealed that querying the cloud machine at runtime was occasionally slow. Amazon guarantees uptime for their servers and while access is typically very fast, various tests showed that querying the S3 bucket added anywhere from 0.5 to 5 seconds to the launch time of the BUI app.

Because we are obsessed with speed and making BUI as fast as possible, adding several seconds to the launch time was unacceptable, especially because new development builds typically occur just once per day. Since the files in the bucket did not change very frequently, constantly querying the bucket every time a user hits the download page would be overkill. If we could somehow query the bucket on a regular basis and save out the results to a local file on the server, then we would shave several seconds off the launch time.

BBj's new scheduling feature fit the bill perfectly! In just a couple of minutes, we set up a job to run a BBj program that retrieved the current list of available cloud builds and saved the information out to a file on the server. We configured the job to run every 15 minutes so the download page would never be out of date by more than a quarter hour. The job itself only takes a few seconds to run so the load on the server is negligible. We then modified the BUI download program to retrieve the list of available builds from the local file on the server, which occurs almost instantly.

The benefits don't stop there. Because we have replicated servers located around the world, the BUI app is served to customers from the server nearest to them. A user in Germany, for example, will be running the download page app from our server in Ireland, while we in Albuquerque get it from a server in California. All of this is transparent to the end user by way of geo-aware servers. Additionally, the replicated servers automatically get the latest set of available builds as the scheduler saves that information to a local file that is automatically pushed to all of the servers via BBj's replication ability. Lastly, Amazon's CloudFront content delivery network sends the desired BBj build to the user from the server closest to them, further streamlining the download process.

Summary

With the addition of a powerful platform-independent scheduler in the BBj product, administrators and developers have more power and flexibility at their fingertips. It is no longer necessary to use one or more third party schedulers to manage BBj backups or other business processes requiring execution at regular intervals. And finally, since the scheduler is built right into the BBj system, it makes certain interactions with BBj-related processes such as replication, much easier to configure and manage. ■



For further information regarding the refactor and optimization of the BUI-based BASIS Product Suite Download page, see *The Anatomy of a Web App Makeover* at links.basis.com/12webapp

Continuous Innovation at BASIS

Building and Testing in the Cloud

B BASIS engineers are responsible for completing a status report template each week. It usually isn't difficult for me to fill in the first four sections and describe what I've worked on or elaborate on problems I've encountered or summarize meetings I attended; that's all pretty standard stuff for a status report. But those last two items - *Interesting Article Review* and *Innovative Ideas* - confront me every week. They are a standing challenge, like a gauntlet tossed at my feet. They are a rebuke for my complacency and resistance to change, a nagging reminder that I need to spend time thinking about new technology and new ideas.

You see, BASIS is constantly on the lookout for better ways of doing things. More than any other company I've ever worked for, BASIS strives to keep abreast of the latest technology and continually improve the software development process. Some of these technical innovations are plainly visible in our products (BUI, anyone?), while others are implemented "behind the scenes" and make us a stronger, more successful company. This article tells the story of one such improvement, something that indirectly benefits everyone who uses BASIS technology.

It Wasn't Broke, but We Had to Fix It

The Business BASIC language, as you might imagine, is a large and complex piece of software. It has thousands of source code files and associated libraries. A few years ago it took an equally complicated system to build it. We appointed a machine as our dedicated build server, a pile of arcane build scripts written in various languages, and a designated "Build Master" engineer who spent all of his time keeping everything running. This arrangement got the job done and worked well for decades. It was clear to everyone, however, that it had several inherent disadvantages.



By Mike Phelps
Software Engineer

- **SLOW** - Getting all the source files from our source code repository, compiling them and assembling them into the final downloadable and installable package took many hours every day. During a normal day, we could get only two builds at most.
- **SERIAL** - Only one build could run at a time. Quite often we needed to publish daily builds of more than one BBj® version at a time, such as a development build (representing the latest work found in the "mainline" version of our source code repository) and a release build (representing a numbered release version of BBj). The build system could not perform simultaneous builds, guaranteeing delays during our most hectic periods.
- **MANUAL** - The build system was not completely automated. A human being always had to press the button to start a build, and if the build failed, a human being (the Build Master) had to comb through the log files to determine the cause. If, late in the afternoon, an engineer checked in a source file containing a syntax error, we wouldn't find out until the morning of the next day that it caused the nightly build to fail. The Build Master had to do some manual detective work to find out what file caused the build to fail and who checked that file into the source code repository so that person could be tasked with fixing the problem.
- **FALLIBLE** - There was always the worrying potential for catastrophic failure. If our Build Master ever happened to get hit by a bus, it would have taken a long time for someone else to learn how to operate and maintain the system. Likewise, had our dedicated build server ever failed, we would have spent several painful days trying to get another machine properly configured to take its place. We had no immediately available backup server or personnel.

Obviously our software build methodology was ripe for a little innovation. Nothing was broken, but we wanted (needed) to fix it anyway, so under "*Innovative Ideas*" I described how '[continuous integration](#)' (CI) could make our build process better.

Learning to Continuously Integrate

The term continuous integration refers to a software development practice where teams of programmers frequently synchronize with and commit their changes to a code base repository, then rebuild the code base after every change is committed. This takes the pain out of getting code from many different people to compile successfully, and helps catch bugs by providing much quicker feedback about build results. Continuous integration is closely associated with other development techniques such as ‘continuous testing’ and ‘continuous deployment,’ where the code is constantly tested and constantly delivered to the end users.

These techniques all imply a high degree of automation, efficiency, and speed; things our build system conspicuously lacked. It all sounded wonderful, but we realized that adopting this technology was going to be a tall order. It was not possible to convert a long-standing, deeply-entrenched-in-the-corporate-culture build system overnight. There were several prerequisite steps that must be taken.

First of all, the code base must be stored in a version control system (VCS) such as CVS, Subversion, Git, Mercurial, or one of many others. CI servers, the software application that takes charge of doing software builds, need a convenient single location from which to get the code to build, and need to work with today’s popular version control systems. BASIS has always used a VCS and currently uses Subversion to maintain the Business BASIC code base.

Next, the software build must be automated, meaning that the code base should compile with some type of script executed by a command from a shell prompt. In its simplest form, a CI server is a constantly running program that listens to signals from a VCS repository. When the VCS announces that a file has been added or deleted or modified, the CI server checks its list of build projects and starts a software build by invoking the command associated with that project. The CI server does not build the software itself; it runs a command that starts the automated script which does the build. When the build is finished, the CI server intervenes once more to store the results in an archive and notify any interested parties about success or failure.

Our existing build system was already partially automated; we had a collection of scripts that we could start from a command-line shell. We had developed these scripts over many years, using whatever scripting technology the build system architects were most comfortable with at the time. We decided to start over from scratch using Ant ([Another Neat Tool](#)), a well-known and extremely flexible build scripting system written in Java. This rewrite effort brought much needed simplicity and consistency to the build process, but it was by far the most time-consuming and difficult part of the conversion to CI. Ant allowed us to store the build scripts in the same directory packages as the code they were meant to build, or in other words, each time it checked the source code out from the repository, the Ant build scripts came with it automatically. Ant was easy to learn and understand, and best of all, Ant was able to compile the BBj code base in less than ten minutes!

With the prerequisites out of the way, we were ready to begin using a CI server for the first time. We chose

[CruiseControl](#), the “grandfather” of CI build servers. We set it up on a spare Linux server and turned it on to building BBj whenever anyone committed a change. These builds were not complete in that they did not produce the installable product delivered to customers; they were meant only to give quick build success/failure feedback to the engineers. The result was fewer failed builds on the main build system.

We later converted to a new CI build server called [Hudson](#) in order to take advantage of its beautiful web-based control system. Instead of arranging for each automated build with hard-to-decipher XML files, we configured the complete system and every build project by filling out GUI forms on a web page. After that, enthusiasm for the CI conversion really began to take off. More engineers got involved in running and maintaining the build system since it was easy to comprehend and fun to operate.

This proved so successful that we gradually added more tasks to the CI build system, like documentation builds. While fleshing-out and expanding the CI build system, we continued to rely on our original build system to produce the final product. Testing involved comparing the CI results with the builds from the legacy system. When the results were the same, we knew we had succeeded.

Ascending to the Cloud

BASIS moved to the cloud in 2010. [Cloud computing](#) refers to the increasingly common practice of purchasing computing power from a commercial services provider instead of maintaining an in-house collection of servers. (The word “cloud” in this term means the offsite collection of computer resources and data storage typically shown in diagrams as a cloud symbol.) We contracted with [Amazon Web Services](#) and began re-hosting our operations from on-premise servers to [Amazon Elastic Compute Cloud](#) (EC2) instances. We moved our Subversion source code repository to a persistent cloud server that summer and then transitioned the build system later in the fall.

Hudson worked extremely well in the cloud, thanks to a special Amazon EC2 plugin developed by Kohsuke Kawaguchi (the original creator of Hudson at Sun Microsystems). Although our ground-based Hudson installation could handle multiple simultaneous builds, the small number of machines in our onsite server lab that we could use as slaves still limited us. During busy times the Hudson server would get progressively slower, while requested builds would pile up in a queue waiting for a slave machine to become free. The Hudson CI server in the cloud, on the other hand, was able to start as many slave instances as required to handle the load. We could do an unlimited number of different simultaneous builds, and then scale back by shutting down the slave instances when their work was finished and the crunch was over.

In early 2012, when the original author of Hudson and most of the project’s developers had a disagreement with Oracle (the corporate sponsor who took over from Sun Microsystems) and decided to fork the code base, we followed them to their new rebranded CI server, called [Jenkins](#). The change was painless.

The use of a CI server and the move to the cloud caused a revolution in our testing and deployment process as well. Jenkins allows “chaining” build projects together in various combinations. This means that we could break up long,



multi-step processes into smaller individual build projects, then link them together so that any given build step has a 'parent' predecessor build project and a 'child' successor build project. The 'child' build projects do not run if their 'parent' builds fail. We wrote Ant scripts to initiate various kinds of tests and then linked them in Jenkins to the software build projects whose results we designed to test. Each time a software build successfully completes, another build starts which retrieves the compiled code from the CI server's archive, assembles it into an install package, and uploads it to an [Amazon S3](#) (Simple Storage Service) 'bucket' in the cloud which anyone at BASIS can find. After the software build and packaging/upload projects complete, the CI server instantiates new slave instances in the cloud and on premises, which correspond to all the platforms supported by Business BASIC, loads each slave with the packaged build results, and calls the associated Ant scripts to initiate testing. The entire process is totally automated and runs without human intervention. (Well, almost. A human being triggers the process by checking in a change to the code base.)

Living in the Future

We are doing things now that were in the realm of science fiction a few years ago.

- The open-source Selenium testing framework integrated very well with our CI server, allowing us to automate testing our BBj GUI interface classes and each individual method belonging to them. This kind of testing previously demanded hours of tedium from a human being sitting in front of a screen with a mouse and keyboard, meaning that it was not done very often.
- At BASIS, we run the administrative and operations side of our company using software written in Business BASIC (we 'eat our own dog food'). We have a development environment in which we test new code and a production environment where we use stable code. In the days before automation and the cloud, getting new code from the development environment

over to the production environment was a somewhat iffy manual operation. The two environments were not identical; what seemed to work in development might not work in production. Now, the issue is resolved with some help from the cloud. When we want to deploy the latest software version to the production server, we invoke an instance of a server that is identically configured to the production server, run tests to verify everything works, and then use [rsync](#) (the Linux remote file synchronization utility) to copy all the new or altered files to the production server. Quick, easy, and as foolproof as it gets.

- Running a TechCon conference used to involve very expensive, time-consuming configuration of in-house and rented computer equipment. All the issues and costs involved with physical transportation, installation, networking, testing, maintenance, and takedown were our responsibility. The cloud has made that old level of effort seem incredible, even ridiculous. We can now prepare [Amazon Machine Images](#) representing a fully configured computer running MS Windows or Linux with all the necessary software pre-installed, then invoke as many instances of them as we want, whenever we want, from the presenter's personal laptop while the demonstration is in progress. We can travel light, but still take with us all the power we need!

Summary

Our new CI cloud-based build, testing, and delivery system is fast, easy to operate, almost infinitely scalable, and (we believe) disaster-proof. The payoff has been remarkable. Is it perfect, or for that matter, is it even finished? I think not, it's a continuous process. We always have a list of improvements to make, inefficiencies to iron out, and new features to implement. With the cloud, there are cost savings available when we tweak this or cut back on that or take advantage of special offers. In this business, nothing remains unchanged for long...except for the weekly BASIS engineering status reports, which still end with the twin challenges - *Interesting Article Review* and *Innovative Ideas*. ■



- For more information on continuous integration, read
 - Continuous Integration - Fowler, Martin. (2006). *Martin Fowler*. Retrieved 6 November 2012 from bit.ly/1k01VP
 - The Cornerstone of a Great Shop – Jared Richardson *Methods & Tools*, Spring 2006 from bit.ly/eXtyT9
- Tools mentioned in this article include
 - Ant ant.apache.org
 - Amazon AWS aws.amazon.com/documentation
 - Hudson wiki.eclipse.org/Hudson-ci
 - Jenkins jenkins-ci.org
 - rsync rsync.net
 - Selenium seleniumhq.org

EMQUE On Cue With BUI Apps That are 'In'



E MQUE's journey into the browser user interface (BUI) world began two years ago as chronicled in the BASIS Advantage article *The Dawning of a New Age with BUI Apps at EMQUE* at links.basis.com/11emque. In their first steps, they wrote two new apps - *The Foreman* and *The Owner* - for use on mobile devices. So now, two years later, we wonder;

"Who is using these apps?"

"What has been the response?"

"In what direction is EMQUE's journey continuing?"

"What sage advice can EMQUE share with others who might (should) follow in their footsteps?"

Since beginning their journey, EMQUE has certainly moved forward with several innovative industry-leading revisions. Their products have risen to the top and are now utilized in several notable New York City renovations, including the Jacob Javits Center and Madison Square Garden.

In the Community

For the second straight year, from May to November during the Knicks basketball and Rangers hockey off-season, Turner Construction is managing the [Madison Square Garden renovation](http://links.basis.com/12toc) project (time lapse at vimeo.com/47174032).



By Susan Darling
Technical/Marketing Writer

Phase 1 ran during the 2011 off-season, Phase 2 completed this November, and Phase 3 will finalize the project in November 2013.

Because of time constraints as well as restricted building use, several trade contractors who are EMQUE customers work around the clock in three 8-hour shifts. These construction foremen use iPads and EMQUE's BUI app, *The Foreman*, to complete their work tickets in this completely paperless solution. The foremen enter the materials manually or download the purchase order information from their in-house servers into the work ticket, type in the description of work, number of hours, and with a tap (iPad's click), create a PDF for the customer to review that even includes their company logo. If the customer requires a signature on the work ticket, the foreman captures that signature in BUI, which adds it to the PDF and then destroys the captured signature as a security measure. In the final step, the foreman selects the recipient's email address and sends the work ticket out instantly.

All of this information is entered in real time so it is available immediately for the accounting team at the home office to process and generate the bills. Now, something that used to take weeks of paperwork processing, completes instantly. Using this mobile app is far more efficient, saving time and minimizing potential errors with real-time entry, day and night. Add the fact that Turner Construction created a free hotspot at the Garden and since iPads use WiFi rather than cell service, there is no additional data overhead. EMQUE's *Foreman* solution is easy, real time, penny-wise...and very attractive!

In the Media

EMQUE's move to BUI has piqued great interest in the industry. In fact, even in a rather slow market, interest has gone very public. And that is exactly what Mike Quagliarello, EMQUE's President and Chief Architect, intended.

"Our approach is that if we get our name out there enough now, when the economy opens up, they'll look to EMQUE because they've seen our name out there."

Advertising, local television, and trade journals have all been vehicles for EMQUE's name recognition and telling their story.

A local television station reported on EMQUE's move to mobility with their initial BUI apps. That report appeared in the Long Island Business News on June 24, 2011 and stirred up great feedback. Watch it at vimeo.com/25554962#.

A high-end national building magazine popular for its contracting information, Engineering News Record (ENR), contacted Quagliarello about his mobile apps and published an article about the apps in their October 22 edition. In November, a corporation that publishes local newspapers nationally, ran an article about mobile apps and turned to Quagliarello for his expertise.

In addition to all that press, this past spring the Long Island Software & Technology Network (listnet.org) awarded Quagliarello and EMQUE the distinguished LISA (Long Island Software Award) at a ceremony attended by 400 technology professionals from around the region. EMQUE earned this award for their cutting edge work in wireless apps for the construction industry, namely *The Foreman*. It offers real-time access to functions vital to maintaining efficiency and productivity on tablets like Apple's iPad. Congratulations, EMQUE! Read their press release at emque.com/LISA_pressrelease.pdf.

In the Field

In the old days of CUI apps, screen layouts and structures were based on that character-based 80x25 world. Quagliarello's move to GUI in 1998 meant more than just putting a pretty face on the app, it meant revisiting the app and solving old business problems using the tools of the new environment. Once in GUI, the move to BUI was simple, but again, they revisited the business process and designed new solutions using BUI and the iPad tablet. By doing so, Quagliarello solved some issues with mobile devices running in BUI that their apps in non-browser-based GUI and CUI could not solve.

EMQUE saw a new opportunity to put timesheet entry into the field. How much more efficient could it be than to enter time clock information real-time, on a smartphone, while in the field? They developed the first version of mobile timesheet for the iPad. Quagliarello notes,

"Initially the cost of iPads restricted us from selling more apps. Some of our clients might have 40 or more users of those initial apps, so the equipment cost was prohibitive in this soft economy. We began looking at how we could produce similar solutions on a smartphone. Once again, we found if we looked at our previous solutions to the problem and 'wiped the slate clean' in our minds, we could

design the solution again, but this time work on a smaller footprint – and we did – our TimeEntry app has raised a lot of eyebrows!"

Adding to *The Foreman* and *The Owner* apps already in BUI, EMQUE has written several specialty mods for *The Owner* such as cost reports, cash flow and income statements, and the ability to view and email aged receivables, among other reports. EMQUE has also moved their Estimating app to BUI and just recently released seven rather large apps for the iPad in what they call the *Project Manager Suite*.

EMQUE is approaching business solutions in their *Project Manager Suite* from a completely different angle. Quagliarello is very excited about cascading style sheets. Compare the look of their Time Sheet app in **Figure 1** and **Figure 2** when applying some prototype CSS magic.



Figure 1. Job Details before and after applying BASIS-supplied prototype CSS



Figure 2. Time Details before and after applying BASIS-supplied prototype CSS

"As the second half of 2012 approached and we found ourselves mastering the nuances of BUI and the various devices we were working with, our attention turned to designing apps that are not only functionally efficient, but also looked cool. We welcomed CSS in BUI with open arms. This is not your father's Business BASIC anymore!"

In the Future

Over the last three to six months, business has been picking up for EMQUE customers. While customers begin to purchase iPads for their foremen, they are migrating to the BUI apps for their laptops. New customers are just now starting to talk to them about *The Foreman* app, so Quagliarello hopes to gain new customers in the next six months. They are certainly seeing growing interest in their BUI apps.

In the Advice Column

The greatest pearls of wisdom are straight from Quagliarello's mouth...

"When we moved to the GUI world in the late 90s, we revisited the business processes that were automated in our character systems. Now as we move full throttle into BUI with new devices, we are building a better mousetrap once again."

"If I was writing this article, I'd title it 'What the h— are you waiting for?' I don't understand why they [other BASIS developers] aren't moving forward... Give up your WYSE50 terminals!"

As EMQUE continues their journey along the technology trail, customers are beginning to line up to use the newest technology and enjoy the freedom that comes with reducing the paperwork jungle. Cost savings, increased efficiency, improved user experience, support for a variety of devices that meet individual needs and budgets, all contribute to making EMQUE's construction management solutions the "in" apps. And if they are developing totally cool "in" apps with BASIS' BUI, they must be doing so as part of the "in" crowd of developers. BASIS technology's backward compatibility means that you are already part way "in." There's room for many more...it's time for you to join in! ■



- Read the first steps of EMQUE's journey in *The Dawning of a New Age with BUI Apps at EMQUE* at links.basis.com/11emque
- Watch the Java Break that featured EMQUE's successes and a demo of the Time Sheet app running the CSS'd version at links.basis.com/jb-emque
- Find more Java Break presentations at links.basis.com/javabreak

EMQUE Consultants, Inc. develops and custom designs, for commercial construction contractors, a fully integrated suite of applications employing best industry practices. Perfect Project, EMQUE's name for the created systems, highlights its seamless integration into an office automation suite. A Perfect Project installation makes financial analysis of business operations a breeze, helping customers know when they are profitable and empowering them to make better financial decisions.

Mike Quagliarello, founder of EMQUE, is President and Chief Architect.

Visit EMQUE at www.emque.com.

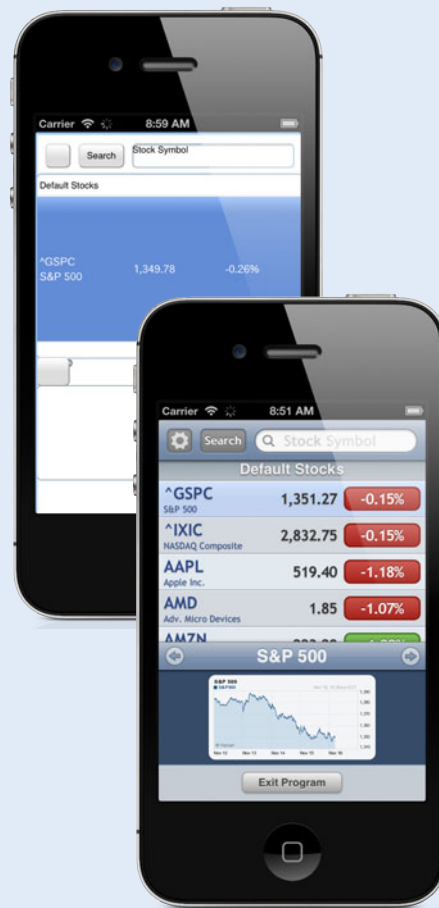
emque

Make Your Web App *Sizzle* With CSS

Attend this all new
training class

May 17th
after TechCon2013

Learn how to
go from this...



...to this!

Register today
links.basis.com/tcreg



A Web Service Sprouts Great Benefits at Bluegrass

For more than a decade, BASIS has propagated the ease and benefits of hosting a Web Service with BBj®. Today, there is still no easier or better way to communicate, share data, and invoke functionality between disparate systems written in different languages and running on different platforms.

So, have you grafted a Web Service into your app yet? Perhaps you haven't considered it or thought of a worthwhile application, or perhaps you have thought about it but put it off until you had stronger motivation.

Take a close look at how one BASIS customer sowed their system with a Web Service that sprouted great cost savings and tremendous increase of productivity.

Look at Bluegrass

Meet Bluegrass Family Health, a managed care corporation, with a tremendous records processing load. With such a high volume comes an even higher need for accuracy in correct coding. To meet that need, Bluegrass chose to implement "ClaimCheck" from McKesson, America's oldest and largest healthcare services company. ClaimCheck is a proven and widely-used comprehensive medical claims coding and review software system that audits claims for correct codes and to assure that the proper payment rates are applied. Bluegrass shared in the industry's regard for McKesson's off-the-shelf ClaimCheck solution as "best-of-breed" and wanted to join the many other insurance companies who use this system.



By Susan Darling
Technical/Marketing Writer

Look to Web Services

While ClaimCheck is a Windows-only product, Bluegrass runs DIAMOND 725 in PRO/5® completely on AIX RS6000 UNIX so they needed a dependable cross-platform solution that interfaced equally well with UNIX and Windows. Web Services does just that, seamlessly using modern network protocols without the more archaic method of hopping files over FTP.

Bluegrass called upon their BASIS reseller EMS, led by President Dave Cominsky and Software Developer, Karin Parker, and together they began looking into the process of implementing ClaimCheck into Bluegrass's enterprise system. The first step was to write the BBj code for consuming the Web Service so they reached out to BASIS for assistance. BASIS Engineer Brian Hipple provided sample BBj code to guide them through the process. Basically, their software solution, DIAMOND 725, would send the claim data in XML format to the ClaimCheck server for verification. Once verified, ClaimCheck would then process the data and generate a response to update the claims in DIAMOND 725.

While they were writing and testing the BBj code, Bluegrass IT Director Preston Gorman was busy defining and customizing rules that the ClaimCheck code review software would follow to validate claims and remediate errors. Gorman explains,

"For example, let's say a maternity claim comes into the DIAMOND system but is actually billed for a man. The extract process pulls all claims and then sends them to the ClaimCheck server to look at the codes and any related rules such as "claim type=maternity" and "gender=male". We had to define the server's response when it identifies this particular claim and the rules for how to respond...whether to reject the claim and request re-submission, whether to deny the claim, or whether to adjust the gender to female."

Bluegrass Family Health, a subsidiary of Baptist Healthcare System, is a not-for-profit managed care corporation based in Kentucky that offers both fully insured and self-funded services to commercial employer groups. Their provider network is made up of hospitals, clinics, pharmacies and doctors. <http://www.bgfh.com>

Bluegrass Family Health

Gorman continues,

"Before ClaimCheck, we used code review software that generated reports of these types of errors for our staff to review manually and decide what to do and hopefully resolve correctly. Now, adding our own rules and intelligence to the huge library of rules 'tried and true' in other larger health plans, we can rest confidently on the thoroughness and accuracy of this process. It's totally automatic...100% hands off."

Accurate and automatic; this new process delivers great cost savings as it ensures correct coding. In addition, it flags bills that might be coded inadvertently with a more expensive billing code or in a manner determined to be unethical or fraudulent.

What has been the impact of consuming this ClaimCheck Web Service? Gorman answers,

"It has helped us a great deal by delivering a simple and effective architecture in which we could interface two very different applications. The Web Service helped us integrate the operation of a BBj environment in a uniform manner with an application built entirely on Microsoft technologies in a virtual environment. The result has been a great improvement in the number of claims that we can audit automatically and accurately."

Look Back

The original scope of the project was to integrate with McKesson's Web Service and to migrate the enterprise entirely to BBj. To do so also required taking the time to test the core components - CRM, billing, eligibility, and batch import/exports, to name a few. This was a huge undertaking that required more time and resources than were available.

As a result, Gorman discovered and now quickly advises others in the BASIS community to identify the scope and stay focused on that goal.

"As we began this project, we lost track of the narrow scope of developing the Web Service interface and deploying it. If we had decided early on to run a dual-environment for a while before migrating the entire environment over to BBj, I think we could have been up so much more quickly and easily."

Look Ahead

While national healthcare is still reforming, Bluegrass is taking a wait-and-see approach before prioritizing their next steps. Currently, their wish list includes moving ahead with migrating their entire enterprise to BBj and implementing some other key Web Services.

One Web Service of great benefit would be an EDI (electronic data interchange) system that provides XML wrappers in addition to the old X12/ANSI format to facilitate delivery of files to their ultimate destination in the proper format. Another great application for Web Services would be a real time connection between claims payment data in their DIAMOND application and their clinical management in another McKesson product. Currently, the clinical management data is batch processed nightly for real data "every 24-hours." A true real-time connection would allow nurses in the field to see up-to-the-minute claims history and clinical management data as they pre-authorize procedures and answer questions about medical care that a patient is about to receive. That patient may have seen the doctor that day for a procedure or filled a prescription moments earlier so the nurse and other health care providers could now make a good medical decision based on accurate and recent clinical activity. This would be a major win for both the patient, who deserves the best healthcare; and the provider, who wants to give the safest and best healthcare possible.

The next step in migrating their full enterprise to BBj would be to move their database from PRO/5 to BBj. While this is a relatively small step, the return is huge. With their data in a BBj database, Bluegrass could immediately access triggers and/or replication to bring that real-time data to their clinical management system, without the need for a third party solution. Simply put, every time data is modified, added, or deleted, a trigger would replicate that change to their clinical management system, keeping them in sync.

Summary

Harvesting a bounty of benefits may just be a simple Web Service away. Regardless of how fertile the environment or recent your BBx® generation, a Web Service may be the answer to taking your application to a point of greater return. While you too may desire a complete overhaul into BBj, take Gorman's advice and keep the scope simple and focused, to begin sampling BBj bounty earlier in the process. Then sit back and enjoy. Happy harvesting. ■



- Read these BASIS Advantage articles for -
 - The basics on writing a Web Service, *From Legacy to Enterprise With BBj Web Services* at links.basis.com/09wsdemo
 - Enterprise Manager's new Web Services tool, *Jetty Offers Legacy Programs via Web Services* at links.basis.com/09jettyws
- Search for more Web Services articles at links.basis.com/adv-ws
- Review BBj and Web Services in the online Help at links.basis.com/bbjwebserv



New Ways to Debug Barista

The Barista® Application Framework now includes several new debugging features that simplify your troubleshooting efforts in your Barista and custom callpoint code. Now armed with the ability to interrupt the process by pressing the [ESCAPE] key, you can debug and dot-step through the code! In addition, you can view the dump to see the contents of the workspace, and start and stop tracing. Lastly, you now have the option to

view the namespaces and the contents of the namespace variables. Read on for more details about these great new debugging features.

Listed under Development in the Barista MDI menu are the four new functions as shown in **Figure 1**; Interrupt Process, View Dump, Start/Stop Trace, and View Namespaces. Launching a standard maintenance form, grid maintenance form, or header/detail form automatically enables these new functions.

Let's take a look at how they work.

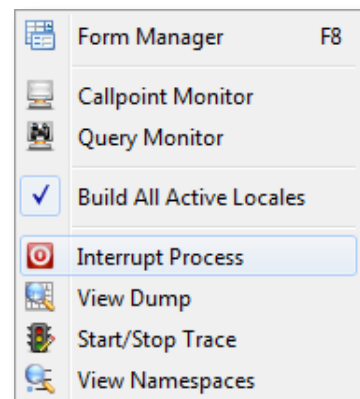


Figure 1. New debug options in Barista's development menu



By Ralph Lance
Software Engineer

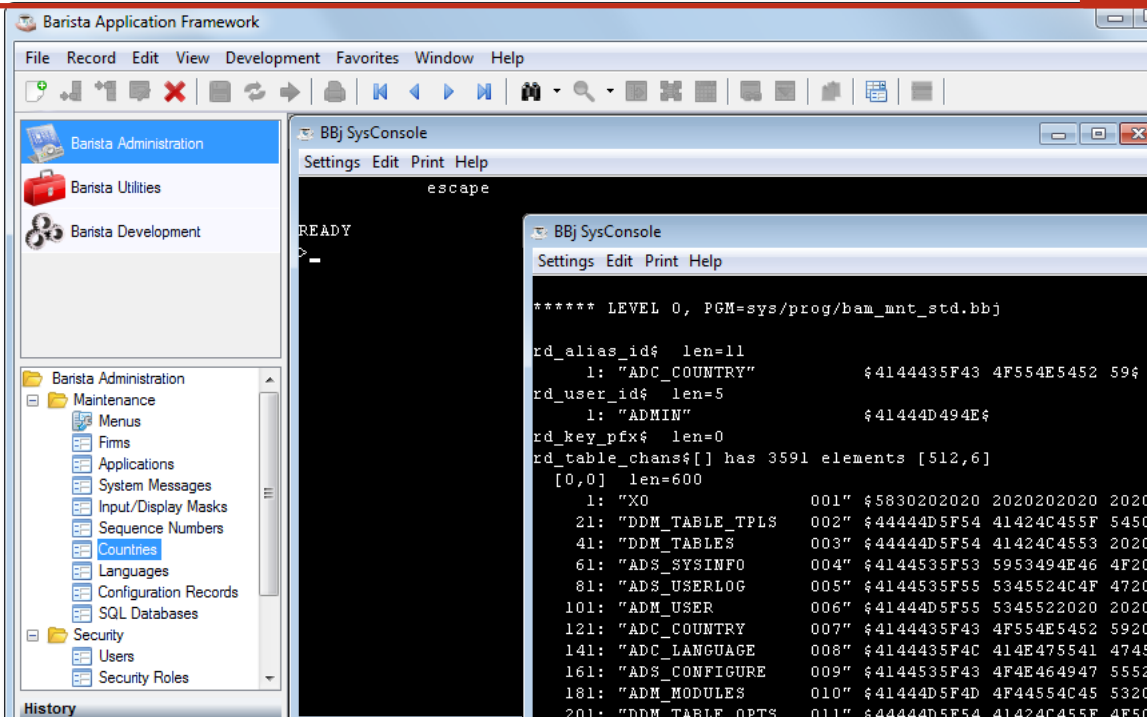


Figure 2. Interrupting a process in Barista

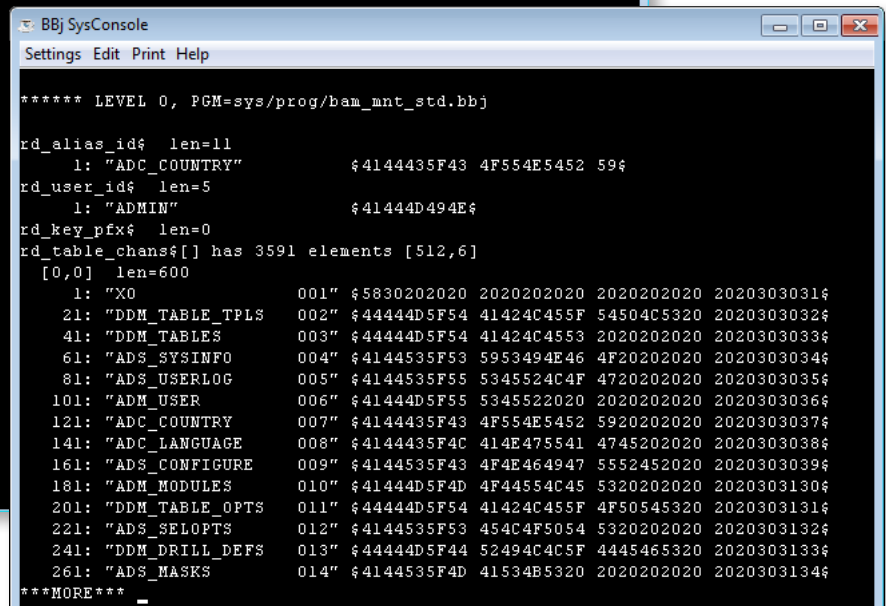


Figure 3. Result of a DUMP command to view the current workspace variables and their values

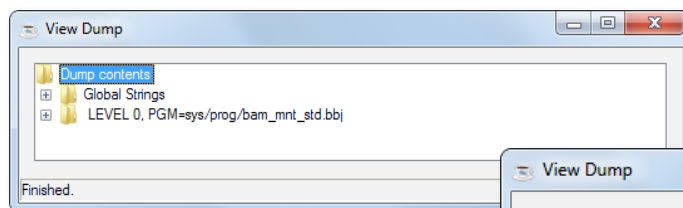


Figure 4. Result of Barista's View Dump menu option

Interrupt Process

Interrupt Process sends an escape to the Barista process as shown in **Figure 2** and then falls to the BBj® SysConsole, if allowed.

From here, you can easily list non-protected programs, interrogate variables, execute dot-step commands, or perform a DUMP command (see **Figure 3**).

View Dump

One alternative to reviewing dump data is to use the menu item View Dump, which presents the data in a developer-friendly tree with global strings separated from the program variable data (see **Figure 4**).

The global strings and program variables appear alphabetically to help find the item in question quickly (see **Figure 5**). Optionally, you can view the individual elements of array variables as well as the template and contents of templated strings.

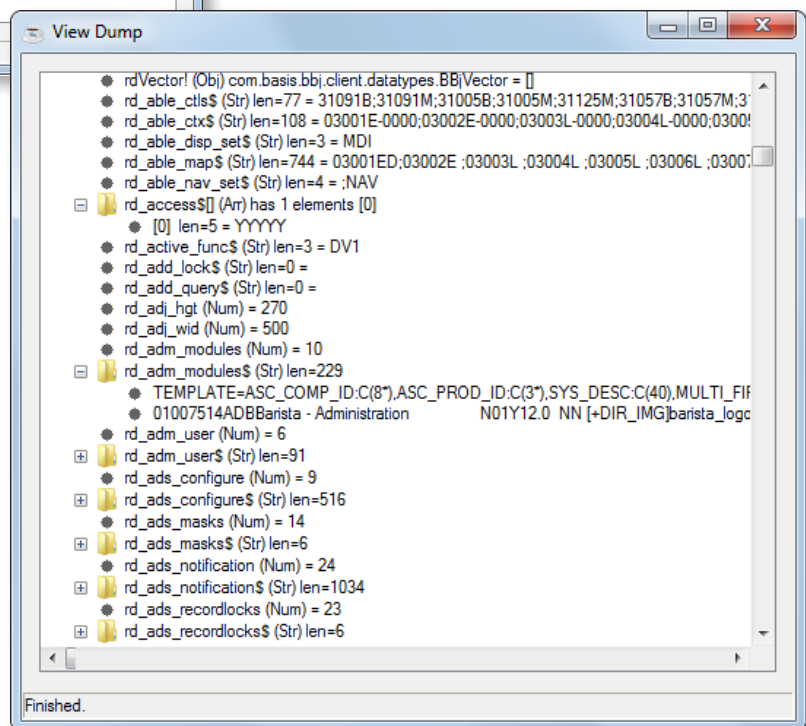
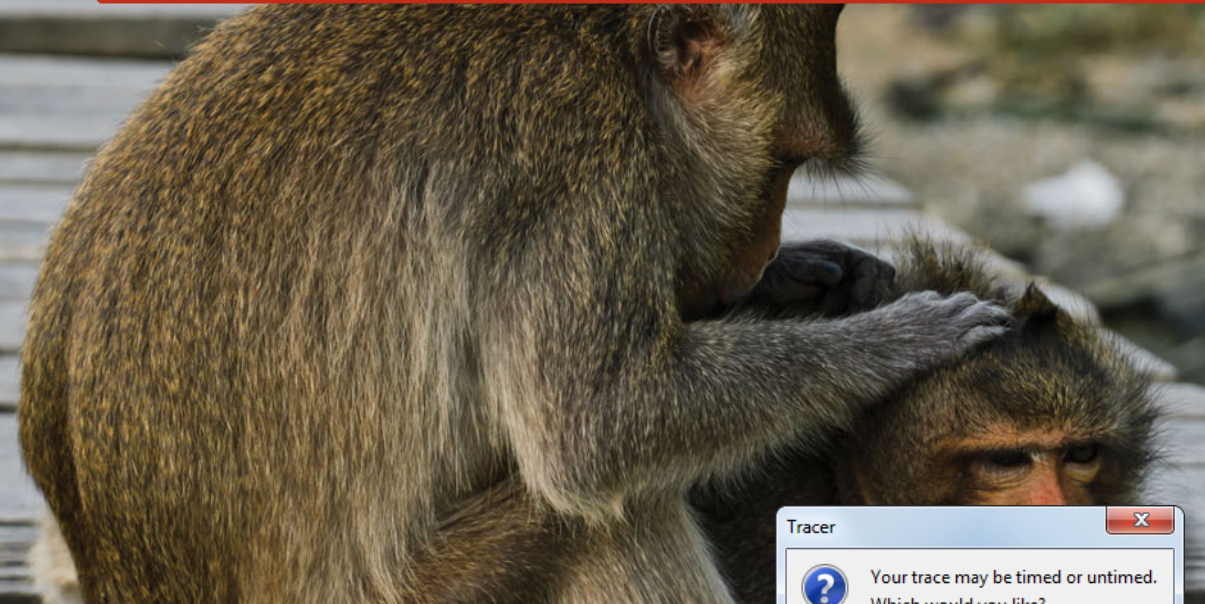


Figure 5. Expanded view of variables



Start/Stop Trace

Starting a trace displays a dialog as shown in **Figure 6** in which you can choose whether the trace should be timed or untimed. The advantage of a timed trace is that you can use the BASIS Performance Analyzer to analyze the program trace file for potential optimizations. An untimed trace is handy for viewing a step-wise path through the code (handy for something like an infinite program loop, for example.) Barista names the trace file using a universally unique identifier (a "UUID" generated for us with a Java function) plus ".trc" and stores it in the Barista `/workarea` directory.

After tracing through the desired processing steps, choose the menu item again to stop the trace and respond to the dialog. If the trace was timed, then you may optionally choose to invoke the Performance Analyzer to immediately analyze the trace file as **Figure 7** shows.

View Namespaces

Similar to the View Dump option, View Namespaces presents a tree view of the GlobalNamespace and GroupNamespace folders, which expands to display the alphabetically sorted contents (see **Figure 8**).

Summary

As robust a tool as the Barista Application Framework is, it is even more powerful and valuable to the developer with its new debugging features. You now have the tools you need to track and monitor your applications as they interact with the Barista Framework. Using Barista is better than ever so add it to your toolset and enjoy the easy and efficient way it can help you be more productive in support of your customers. ■

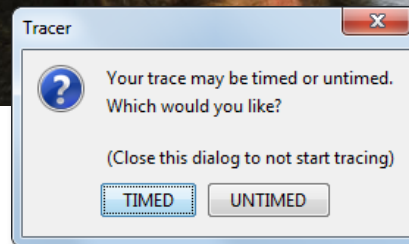


Figure 6. Starting a trace

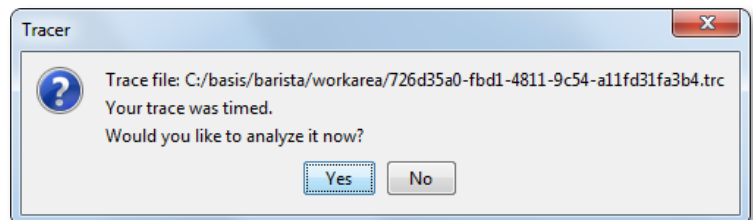


Figure 7. Finishing the trace

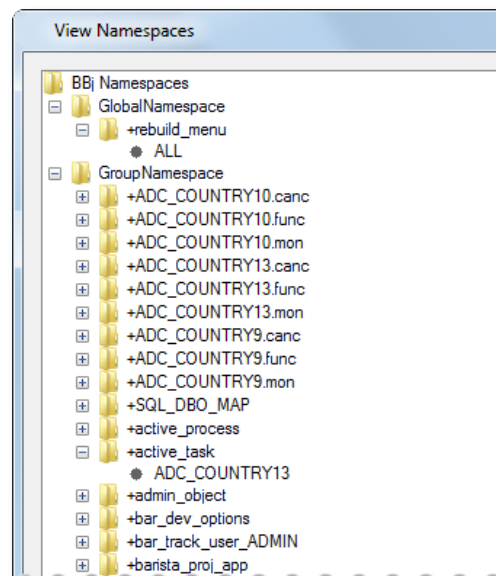


Figure 8. Result of Barista's View Namespaces menu option



- Read *Tuning the Performance Analyzer* at links.basis.com/05tuning
- Find additional Barista resources at links.basis.com/baristaref



Customizable Mobile Report Viewer

BBJaspeViewer Becomes Highly Customizable and Goes Mobile

If you are displaying a report in a BBJ® application, then you have probably taken advantage of the BBJaspeViewer functionality. The BBJaspeViewer is a utility that allows end users of BBJ applications to view and interact with the JasperReports engine, an open source Java reporting library. Reports designed with the WYSIWYG iReport open source, cross-platform report designer rely on the JasperReports engine to generate the report. BASIS built the original BBJaspeViewer, found in earlier versions of BBJ, around a Jasper Java class. This became problematic when BBJ introduced the browser user interface (BUI) because it could not utilize Java client objects in this environment. To overcome this limitation, BASIS rewrote the BBJaspeViewer utility in pure BBJ code and released it in BBJ 12.

Can BUI use Java objects? On the server, yes, on the client, no! Java code requires a Java Virtual Machine (JVM) so without it running on the client, Java cannot run. The client only needs a browser to run BBJ code in BUI since one of the great benefits of

BBJ is that it translates code into the browser-rendered HTML, JavaScript, and CSS. All smartphones, tablets, and other mobile devices include a browser, thus making BBJ a truly multi-platform, “write once run anywhere” language. It was therefore only logical to use BBJ to create a BUI compatible viewer for JasperReports. Since BBJ is an object-oriented language, it was easy to translate much of the Jasper Java Viewer object-oriented code into the BASIS language. Much of this code deals with displaying the report in the viewer. The BBJaspeViewer obtains a PNG image for the report from the Jasper API, translates it into a BBImage via the BBImageManager and then subsequently displays it using a BBImageCtrl. The end result is a perfectly displayed report in your browser that is fully interactive, as shown in **Figure 1**.



By Brian Hipple
Quality Assurance
Supervisor



Figure 1. BBJaspeViewer running in a web browser



Not only does this new BBJasviewer utility version retain all existing functionality, BASIS added new features and functions as well as an enhanced user interface. More control over the viewer is now possible with new methods such as being able to set the initial zoom rate and page number. A new tool button saves the current report page as a PNG image to better share information from the report with others. For example, a department manager reviews a sales report and sees a transaction that is important for the owner of the company to review. Now what? The manager can simply create an image of that particular page and send it by email as an attachment.

To improve the user experience, BASIS also updated the user interface, shown in **Figure 2**, to include a new child window that houses the various tool buttons - updated with new graphics - that manipulate the report.

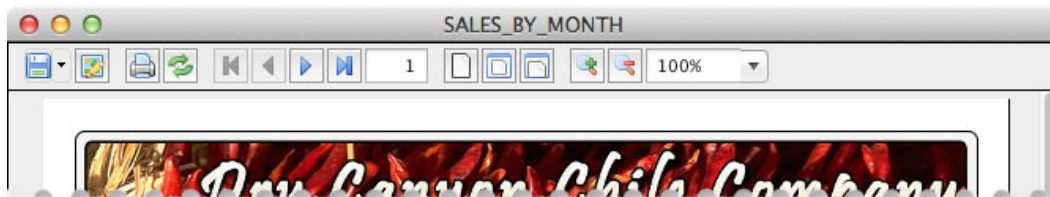


Figure 2. New user interface

Programmatic access to these controls is now possible, making the viewer highly customizable by BBj applications. One particular BASIS customer has customized their viewer by removing the ability to save the report to Google Docs from the [Save] BBjMenuButton. In addition, developers can add any BBj control to the toolbar child window, further customizing the viewer to meet the needs of their application. For a really useful example, have a look at **Figure 3**, which shows the code that adds custom email and fax toolbuttons to the BBJasviewer toolbar and sets the appropriate callback routines. The developer then adds the backing code to send the current report to one or more email addresses selected from a list and the modification is complete!

```
rem Create the viewer window to display the report
viewerWindow! = new BBJasviewerWindow(report!)
viewerControl! = viewerWindow!.getViewerControl()

rem Add tool buttons to the viewer control
viewerControl!.addToolBarSpacer()
emailToolButton! = viewerControl!.addToolBarButton("email.png","Send Email")
faxToolButton! = viewerControl!.addToolBarButton("fax.png","Send Fax")

rem Register callbacks
emailToolButton!.setCallback(BBjToolButton.ON_TOOL_BUTTON_PUSH,"OnEmail")
faxToolButton!.setCallback(BBjToolButton.ON_TOOL_BUTTON_PUSH,"OnFax")

rem Show the viewer window
viewerWindow!.setReleaseOnClose(1)
viewerWindow!.show(0)

rem Process the events
process_events
```

Figure 3. Adding the custom email and fax toolbar buttons to the BBJasviewer window

In addition to these enhancements to the BBJasviewer, BASIS enhanced the BBJas utility as well. Beginning in BBj 13 and higher, the utility adds the ability to fill a report with data from a ResultSet, making it possible to use and display Jasper reports in applications without using SQL or SPROC.

The code in **Figure 4** is an example of this new feature.

```
rem Create a BBJasperReport with result set

rem Use statements
use ::bbjasper.bbj::BBJasperViewerWindow
use ::bbjasper.bbj::BBJasperViewerControl
use ::bbjasper.bbj::BBJasperReport
use java.sql.ResultSet

rem Set the report file
reportFile$ = System.getProperty("basis.BBjHome") + "/utils/reporting/bbjasper/chileco_customers.jrxml"

rem Create the RecordSet
template$ = "CUST_NUM:C(6):LABEL=CUST_NUM;FIRST_NAME:C(20):LABEL=FIRST_NAME;LAST_NAME:C(30):LABEL=Last;"
template$ = template$ + "COMPANY:C(30):LABEL=Company;BILL_ADDR1:C(30):LABEL=BILL_ADDR1;"
template$ = template$ + "BILL_ADDR2:C(30):LABEL=BILL_ADDR2;CITY:C(20):LABEL=CITY;"
template$ = template$ + "STATE:C(2):LABEL=STATE;COUNTRY:C(20):LABEL=COUNTRY"
recordSet! = BBJAPI().createMemoryRecordSet(template$)

rem Add records to the record set
data! = recordSet!.getEmptyRecordData()
data!.setFieldValue("CUST_NUM","000001")
data!.setFieldValue("FIRST_NAME","Brian")
data!.setFieldValue("LAST_NAME","Hipple")
data!.setFieldValue("COMPANY","BASIS")
data!.setFieldValue("BILL_ADDR1","5901 Jefferson")
data!.setFieldValue("BILL_ADDR2","")
data!.setFieldValue("CITY","Albuquerque")
data!.setFieldValue("STATE","NM")
data!.setFieldValue("COUNTRY","USA")
recordSet!.insert(data!)
data! = recordSet!.getEmptyRecordData()
data!.setFieldValue("CUST_NUM","000002")
data!.setFieldValue("FIRST_NAME","Dr. Kevin")
data!.setFieldValue("LAST_NAME","King")
data!.setFieldValue("COMPANY","BASIS")
data!.setFieldValue("BILL_ADDR1","5901 Jefferson")
data!.setFieldValue("BILL_ADDR2","")
data!.setFieldValue("CITY","Albuquerque")
data!.setFieldValue("STATE","NM")
data!.setFieldValue("COUNTRY","USA")
recordSet!.insert(data!)

rem Get the ResultSet from the BBjRecordSet
resultSet! = recordSet!.getJDBCResultSet()
resultSet!.beforeFirst()

rem Params
params! = new java.util.HashMap()

rem Create and fill the report with the result set
report! = new BBJasperReport(reportFile$,resultSet!,params!)
report!.fill()

bbjasperViewer! = new BBJasperViewerWindow(report!)
bbjasperViewer!.show(1)
end
```

Figure 4. Using a `ResultSet` to construct a `JasperReport`

Summary

What started out as a rewrite to get the report viewer to work in the BUI environment has turned into so much more. BASIS now makes it easier than ever to incorporate your application's custom reporting needs, no matter what environment your application runs in. What are you waiting for? Realize the power of BASIS' BBJasper application building block utility today, available in preview beginning with BBj 12.10 and in full release in BBj 13! ■



- Review these features in the online docs at links.basis.com/basishelp
- Read these BASIS Advantage articles
 - *Jazz up Your Applications - Seamlessly Embed JasperReports* at links.basis.com/09jasperreports
 - *New BBJasper Output Types, Including the Cloud!* at links.basis.com/11bbjasper
- Download the sample code referenced in **Figure 3** at links.basis.com/12reportviewer-code

BASIS IDE in Java 7th Heaven



Did you know the BASIS IDE has been updated to support Java 7? Oracle has announced that support for Java 6 will cease in Q1 of 2013 (www.oracle.com/technetwork/java/eol-135779.html) so the IDE is ready for 7! Are you ready?

In this article, we'll review one of the most apparent changes – full support of code completion for Java 1.7-specific objects. Java version 1.7 introduced a new class, `java.util.Objects`, with static utility methods for manipulating Java Objects.

Figure 1 shows the beginning of a BBj® program that calls out this new Java class. The `USE` statement makes `java.util.Objects` available to the BBj code, without errors appearing in the editor.

When you reference Java classes in your code, the code completion popup window connects to a new database that knows all about the latest additions to Java 7. Code completion is triggered by typing the '.' period character after the name of an object variable or a Java class. The popup window presents a list of all the members and methods available to that class, including ones you may not have seen before. Simply select the one you want with the mouse or arrow keys, then click the mouse or press [Enter] and see the method appear in the editor at the cursor position.



By Mike Phelps
Software Programmer

REM BBj Program that invokes a Java 7 class

use java.util.Objects

Objects.requireNonNull(x!, "x! should not be null!")
Objects.

```
java.util.Objects
int compare(Object, Object, Comparator)
boolean deepEquals(Object, Object)
boolean equals(Object, Object)
int hash(Object[])
int hashCode(Object)
Object requireNonNull(Object)
Object requireNonNull(Object, String)
String toString(Object)
String toString(Object, String)
Class class
```

Figure 1. Code completion popup for the Java 7-specific `Objects` class

Code completion is a significant memory and time-saving tool: There is no need to turn away from your work to consult documentation on a web page or in a book. Code completion puts the information you need right inside the editor in the most convenient way possible. The BASIS IDE ships with code completion databases for Java and all the classes in the BBj API and provides code completion assistance for both languages.

If you are still running version 1.6, jump ahead of the game and update to 1.7 now before the clock runs out. BASIS has made the upgrade so you can do so without any last minute crisis. Go to Oracle's site and upgrade today! ■



Download Java 1.7 JDK from Oracle at bit.ly/bMkbpo



EMS Prescribes BUI to Reduce Healthcare Expenses

Rising costs are plaguing most industries across the country. Even in the healthcare world, with reform around the corner, the need to reduce costs is growing as fast as the unknown future that looms over the horizon. For one insurance payor, the BASIS browser user interface (BUI) was the clear answer, providing a solution that not only accomplished their goals of accessibility, but helped deliver tremendous cost-cutting savings.

PATIENT

- A large state healthcare department

VITALS

- Works to assure access to quality health services at all levels of need and life stages
- Estimates they will serve nearly two million people sometime during their lives
- As of July of 2011, serves well over 360,000 people across the state through 390 community agencies, 84 private hospital inpatient units, 157 community residential programs, 674 adult care facilities, and 89 adult family homes
- Runs the DIAMOND 725 solution written in PRO/5® with about 200 concurrent users

SYMPTOMS

- Providers were unable to determine patient eligibility before rendering services
- Treatment delayed or potential costs incurred under dispute for both the State and the Provider



By Susan Darling
Technical/Marketing Writer

TREATMENT PLAN

Reducing rising costs in the healthcare industry is generally out of reach, but this payor saw that they could reign in the costs by not paying the overbilled health care services. While enforcing the contract limits in their process, they also wanted to empower providers with a way to verify eligibility before rendering the services, resulting in reduced billable services and saving scores of dollars. All achieved through the provision of up-to-date information to the provider at the point of delivery of the services.

DIAGNOSIS

Looking at their Treatment Plan, the solution needed to be

- Accessible via the Internet
- Cross platform
- Resource efficient
- Easy to maintain
- Intuitive for users
- Easy to implement/train
- Cost effective
- Technologically compatible with their current hardware and software, DIAMOND 725 built on PRO/5

PRESCRIPTION

EMS Healthcare Informatics President and CEO Dave Cominsky and Senior Software Engineer Karin Parker, along with the State agency's IT director, attended BASIS' TechCon2011 where the browser user interface debuted. A few months later, at the DIAMOND Users Group, the three again saw BUI in action and learned more about its implementation. When the IT director approached EMS about their growing service limitation need, discussions began. Parker explains,

"The idea came to mind of a web portal that we had heard about when we attended TechCon. I was not familiar with GUI development but it was easy enough to harvest the Business BASIC logic from our business app."

With revived faith in DIAMOND 725, EMS resuscitated the application by writing and deploying it in BUI, providing access via a provider portal to view claim history. For the sake of user familiarity and ease of implementation, Parker designed the BUI screen to look similar to the CUI screen shown in **Figure 1**.

test.HAW - HyperACCESS

File Edit View Properties Transfer Automation Window Help

DSPBN Display Benefit Accumulators

—Identification Information—
 Subscriber # : 1008972 Person # : 00 As of date : 06/30/2012
 LASTNAME FIRSTNAME Auths?: N

—Eligibility Information—
 Start date : 08/01/2011 End date : / / Status : E
 Group : FR FRANK Plan Code : DFMCD25
 Benefit package : BBB00001 DEFAULT

—Benefit Rule Summary—

Rule ID	Description	UQT	Basic Amt	Accum Amt	Auth Accum	Remain Amt
MHACTHNC	LIMIT ACT/HB	1.00	0.00	0.00	0.00	1.00
MHASSMTNON	ASSMT-NON PH	4.00	0.00	0.00	0.00	4.00
MHASSMTPHY	ASSMT-PHYS L	2.00	0.00	0.00	0.00	2.00
MHCOUNSEL	FY COUNSELIN	52.00	0.00	0.00	0.00	52.00
MHPHARMGT	FY PHARM MGM Q	24.00	0.50	0.00	0.00	23.50
MHCPST	FY CPST LIMI F	6.00	6.00	0.00	0.00	0.00
		0.00	0.00	0.00	0.00	0.00

<Up/Down>=scroll,<Enter>=display claims,<F8>=Expand Fields,<Home>=exit:

Figure 1. CUI version of the claim history

Extending DIAMOND 725 to a BUI solution breathed new life into the legacy product and handily met the original requirements –

- Deliver cross-platform browser access
- Preserve existing business rules
- Easily deploy with a familiar and intuitive interface
- Use current BASIS technology available at no additional charge making it cost effective (current on software maintenance) + containing costs with a small amount of GUI programming required to deliver the solution

COURSE OF TREATMENT

With the PRO/5 DIAMOND application running IBM AIX on the database or production server, EMS set up a second server with Linux to run BBj®, delivering GUI as an application layer with the business logic they harvested ‘as is’ from PRO/5. The PRO/5 Data Server® became the gateway to access the production data on the AIX server from BBj. A third server hosted LDAP and Web Server tasks to deliver the face of the application via BUI to the providers’ desktops and mobile devices. **Figure 2** illustrates

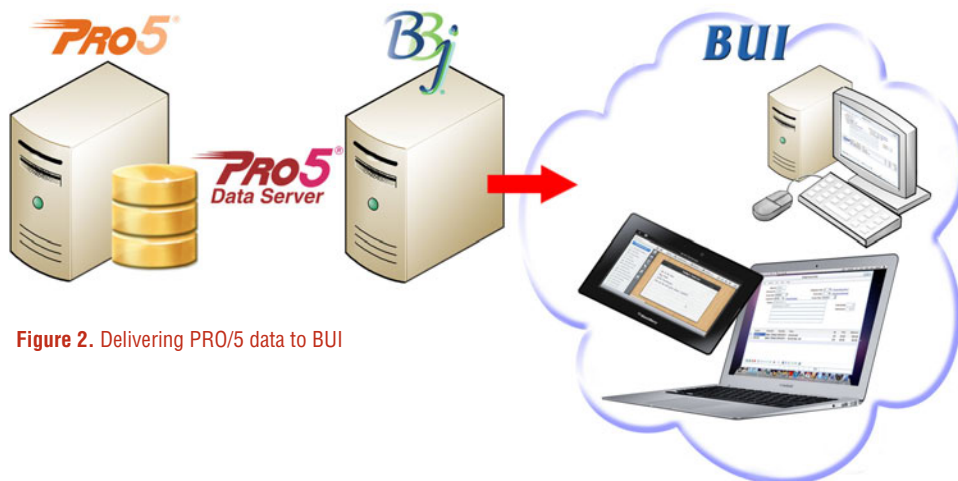


Figure 2. Delivering PRO/5 data to BUI

this configuration. BASIS LDAP support allowed EMS to write a small user validation front end so when providers go into the portal, they are prompted to log in before the one-page BUI app launches. Now with very minimal training, they can look up a name or subscriber number to view claim history and verify if a service would be covered before ever rendering it.

The beauty of BUI is that providers can easily log in from any browser-compatible device to the portal as shown in **Figure 3** and do so without having to install any JVMs or additional third party programs on their device. According to Parker,

“They wanted to use Internet Explorer 8 so we accommodated them and added the Google plug-in, and also have it running for use in Mozilla FireFox and Google Chrome.”

While BUI also gives EMS the CSS functionality to one day change the look and feel of the application, they felt delivering screens similar to the look of the legacy CUI app maintained a familiar user experience and met their goals of “intuitive” and “easy to train/implement.”

TREATMENT RESULTS

Since the BUI app uses data from the actual adjudication database, the healthcare providers now have up-to-the-minute service eligibility information, not day- or week-old information. Providers in any location have access to summarized patient service history and can make informed decisions regarding future authorizations. In addition, by determining which patients need pro-active billing, the providers will see an increase in their cash flow.

From a licensing perspective, the BUI deployment only required an additional 25 concurrent user licenses as no more have actually logged in at any one time, even though there are 500 active provider accounts. Only a single production license was needed to power up to 200 PRO/5 sessions and the providers’ browser access.

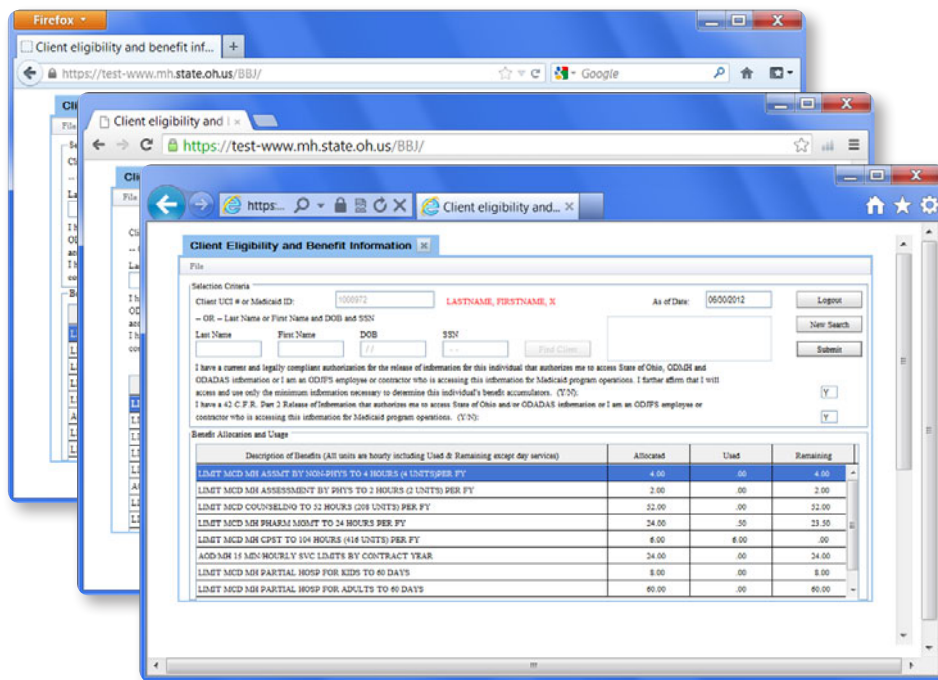


Figure 3. BUI screens in each of the browsers; Mozilla Firefox, Chrome, and Internet Explorer 8

BUI the wonder drug was truly the right Rx. As Cominsky recaps,

"The State was establishing critical service limitations for the first time and had to have a way for the doctors' office to access real-time decision making information. The BASIS team was able to guide us through the process of using state-of-the-art technology that meshed Internet connectivity to legacy PRO/5 data. The resulting BUI app was well received and completely relevant to the success of the State initiative."

TIME FOR A HEALTH CHECK?

If you have a similar diagnosis requiring cost savings, greater efficiency, and improved productivity, then perhaps BUI could be the very prescription to ensure the continued health of your business application. A pain free, minimally disruptive BUI procedure could be just the plan for the recovery of your fiscal health. Check it out today! ■



Watch the Java Break that featured this success story in greater detail and see it in action at links.basis.com/jb-ems

EMS Healthcare Informatics, based in Clarence, New York was founded in 1996. EMS provides middleware applications along with programming and implementation services to the healthcare industry. More than 1,000 users run their solutions and EMS supports several major customers running DIAMOND 725, written in PRO/5 the late 1990's.
www.emscorp.biz



Make Your Web App Sizzle With CSS

Attend this all new training class

May 17th
after TechCon2013

Learn how to
go from this...



...to this!

Register today
links.basis.com/tcreg

BASIS SQL Gets Even Better

As databases evolve over the years, it often becomes necessary to add SQL functionality. While SQL is a relatively standard language for querying databases, various DBMS vendors often add syntax to their SQL not supported by all databases. BBJ® now boasts new SQL support for several SQL features supported by other databases, as well as significant improvements to the way it handles SQL views.

Execute SQL/MySQL Script

The Enterprise Manager now allows an administrator to execute an SQL script on any BBJ database. Script files should be plain text and contain one or more SQL statements, terminated by a semicolon and a new line. Statements can include CREATE and DROP statements for TABLE, VIEW, PROCEDURE, INDEX, and TRIGGER as well as CALL, INSERT, UPDATE, DELETE, ALTER, GRANT, and REVOKE statements. To execute a script, click the button shown in **Figure 1**, located on the database Information tab.

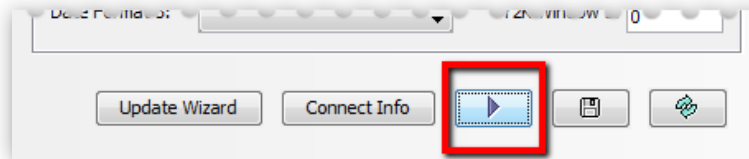


Figure 1. Execute an SQL script file

This powerful feature also includes support of most MySQL syntax for creating, modifying, and populating tables so administrators can easily import MySQL database exports into a BBJ database. When importing from a MySQL script, set the "CREATE TABLE File Type" to ESQL as shown in **Figure 2**.

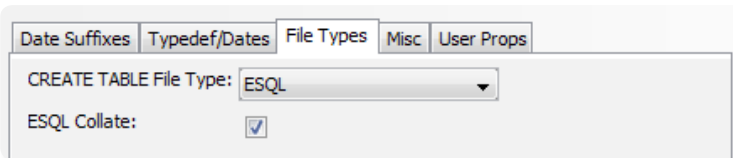


Figure 2. CREATE TABLE File Type Setting

ENUM Type

BBJ 12 now supports the use of the ENUM type ([Enumerated Type](#)) when creating a table. Note that this feature only works with ESQL tables as it relies on ESQL files' constraint capabilities. An ENUM type column's values are restricted to a set of explicit values specified at the time of table creation. For example, the following statement creates a table with an ENUM type:

```
CREATE TABLE my_table (id INTEGER PRIMARY KEY, gender ENUM ('M', 'F')) ESQL
```

This next statement generates an error because 'Q' does not meet the criteria for the enumeration:

```
INSERT INTO my_table VALUES (1, 'Q')
```

This statement will succeed:

```
INSERT INTO my_table VALUES (1, 'M')
```

The new ENUM type guarantees data integrity as only valid information may be written to the database.

REPLACE Statement

The REPLACE statement works just like INSERT with the exception that if the value specified for the primary key in the REPLACE statement already exists, it simply replaces the existing record with the new record specified in the REPLACE. For example, if ID is a



By Jeff Ash
Software Engineer

primary key, the first statement would insert a new record because it is new, while the second would change the NAME to John.

```
REPLACE my_table (id, name) VALUES (10, 'Jeff')
REPLACE my_table (id, name) VALUES (10, 'John')
```

Using REPLACE is a great timesaver for developers, as previously this would require one of the following multi-step processes to accomplish the same thing.

1. Execute a SELECT statement to determine if the record already exists
 - If the record exists, execute an SQL UPDATE statement
 - If the record does not exist, execute an SQL INSERT statement
2. Attempt to insert the record by executing an SQL INSERT statement, trapping for an error
 - If the SQL INSERT failed because the record already exists in the table, execute an SQL UPDATE statement

DATEDIFF Scalar Function

The DATEDIFF scalar function returns the number of days between two specified dates. Parameters can be literal expressions or the name of a date type column; it subtracts the second parameter from the first parameter. For example, the following returns a value of 7:

```
SELECT DATEDIFF('2012-08-08', '2012-08-01')
```

Reversing the order of the parameters returns -7:

```
SELECT DATEDIFF('2012-08-01', '2012-08-08')
```

This simple-to-use scalar function makes it very easy to perform this common date operation without the necessity for writing any application code. Further, using a scalar function to perform calculations makes it possible to use the SQL statement in reports without the need for custom report scripting.

Full Featured Views

Full featured views are a tremendous improvement over the way BASIS SQL traditionally supported views. Prior to full featured views, SQL views were limited to a SELECT column list, list of tables, and a limited WHERE clause. ORDER BY, LEFT JOIN, GROUP BY, etc. were unsupported. However, full featured views support all valid SELECT queries regardless of size or complexity and were first available in BBj 12. By default, adding a database to BBj configures that database to create full featured views. **Figure 3** shows the setting for enabling full featured views.

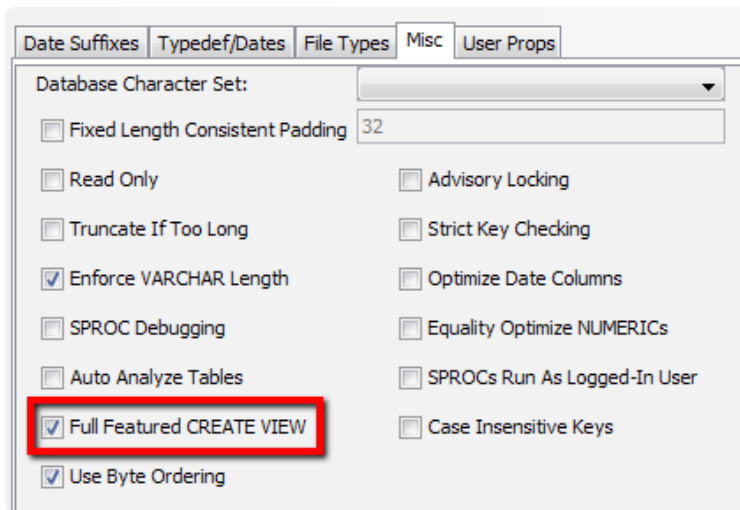


Figure 3. Enabling Full Featured Views in Enterprise Manager

The following example of a complex CREATE VIEW statement now works in BBj with the addition of full featured views, it demonstrates the power and flexibility of this new capability.

```
CREATE VIEW my_great_view AS
SELECT t.name,
       count(t.order_num) AS NUM_ORDERS
FROM   (SELECT trim(c.last_name) + ', '
        + c.first_name AS NAME,
        o.order_num
        FROM   customer c
        LEFT JOIN order_header o
              ON c.cust_num = o.cust_num) t
GROUP BY t.name
ORDER BY t.name desc
```

Summary

All currently maintained database management systems evolve and grow over time to include new features, new syntax, and improvements to existing functionality. Because BASIS is always listening to customers for ideas about improving our products, count on more improvements and new features to the BASIS toolset. The addition of ENUM, REPLACE, DATEDIFF and full featured views gives BBj developers several more tools to use when developing database applications. Please keep the ideas coming! ■





Easily Install Your Apps With the BCI



By Brian Hipple
Quality Assurance
Supervisor

The BASIS Custom Installer (BCI) continues its evolution towards a fully-featured installation program, not only for BASIS products, but for customers' applications as well. It is very appropriate that the word 'custom' is included in the name of this installer. BASIS wrote the Installer with a great deal of flexibility for the developer and it is completely configurable in all aspects of installing any BBx®-based application. More explicitly, the developer can determine which commands to invoke and under what conditions, what files to install, and which BASIS installation wizards to run and in what manner. All of this functionality is specified in either an installation response file or an instructional XML file. These files include new options that BASIS incorporated into the latest version of BBj®.

Recently, the BASIS marketing team came to the engineers with an idea for a campaign to promote our ERP product written in BBj – AddonSoftware® by Barista®. They wanted the user to be able to download, install, configure, and run AddonSoftware without *any* interaction. The intended audience for the promotion would likely be unfamiliar with BASIS or our products, so it was imperative that this was a simple one-click process. They did not want any prompts to appear requiring users to make otherwise typical selections – the Java version, install directory, license registration, license manager, or how to configure and start the BLM or BBj Services. The team wanted it all to work and be configured automatically on all BBj-supported operating systems. Was the BCI up to the task?

Most definitely the BCI could handle the challenge, however, along the way we found opportunities for improvement that not only met the requirements of our marketing team, but also would offer great benefits to the BASIS community at large.

Enhancement #1 - Progress Mode

The BCI gives the choice to configure the installation wizards to run either interactively or silently via the response file. Since we definitely did not want the installation to be interactive, we chose the silent option. What we found out with this option was that depending on the system, it could take several minutes for BBj and AddonSoftware to be installed, without giving users any sense of progression. Here was the first opportunity for enhancement. We added a new progress mode to the response file to display a progress screen in both GUI and CUI. This new option provides the end user an indication of the progress of the installation and improves the user experience. An example of the new progress mode is shown in an excerpt of a response file in **Figure 1**.

```
#####
# Wizard Settings
#
# The following variables set whether or not to run various BASIS
# installation and configuration wizards after the installation of the software.
# Setting a value to [interactive] will cause the specified wizard to be run
# interactively. Setting a value to [silent] will cause the specified wizard to
# be run silently. Setting a value to [progress] will cause a progress screen
# to be shown while the wizard is being run Setting a value to [off] will
# prevent that wizard from being run. The UAC wizard will only be run on Windows
# machines in which UAC is enabled. The license selection and finish wizards
# can not be run silently.
#
# The following value can be [interactive] [silent] [progress]. The default
# is [interactive].
INSTALL_WIZARD=progress
# The following values can be [off] [interactive] [silent] [progress]. The
# default is [off].
UAC_WIZARD=progress
LICENSE_SELECTION_WIZARD=off
LICENSE_REGISTER_WIZARD=off
LICENSE_INSTALL_WIZARD=off
BBJ_BRAND_WIZARD=progress
BLM_CFG_STARTUP_WIZARD=progress
BLM_START_STOP_WIZARD=progress
BBJ_CFG_STARTUP_WIZARD=progress
BBJ_START_STOP_WIZARD=progress
EM_WIZARD=off
FINISH_WIZARD=off
```

Figure 1. Example of the response file that sets the progress mode for the wizards

Enhancement #2 - License Installation

Another ability we needed was to have a temporary license installed by default. To accomplish this, the installer now can utilize a BASIS license file in the installable jar file when it's named **basisdemo.lic**. When found, this license installs automatically and BBj and the BLM configure appropriately to utilize it. You can easily add a license file to an existing installation jar via the syntax and example shown in **Figure 2**.

```
<JAVA_HOME>/bin/jar -uf <BASIS Installable.jar> basisdemo.lic

/usr/local/java/bin/jar -uf
BBjBaristaAddonIDE1201_08-13-2012_1245.jar basisdemo.lic
```

Figure 2. Generic syntax (top) and example (bottom) command for adding a demo license to an installation jar

Enhancement #3 - Platform Specificity

We needed more granular specifications for the Java directory, installation directory, and configuration options. Response file values were not OS specific and only gave the developer one chance to set them, regardless of the platform. Therefore, we added **_WIN** and **_NON_WIN** suffixes to the desired property keys for Windows- and non-Windows-specific operating systems. Now, we can include the options for different platforms conveniently in the same response file. **Figure 3** shows some of the OS-specific property keys.

```
#####
# BLM Configuration Startup Wizard Detail Settings
#
# The following properties are for Windows OS
# The following value can be [service] [login] [manual]. Default is [service].
BLM_CFG_STARTUP_TYPE_WIN=service
# The following value can be [auto] [manual] [disabled]
BLM_CFG_STARTUP_SERVICESTARTUPTYPE_WIN=auto
# The following property is for non-Windows OS
# The following value can be [init] [manual]. Default is [init].
BLM_CFG_STARTUP_TYPE_NON_WIN=init
```

Figure 3. OS-specific property keys for setting specific values

Enhancement #4 - Auto Execute

Lastly, we needed to run the AddonSoftware application after all the components had been installed. This was previously possible by specifying a BBExec node in the instructional XML file: **custominstall.xml**. However, we thought this might be an option that our customers would frequently want to take advantage of so we added this ability to the more accessible response file as shown in **Figure 4**.

```
# The following values can be specified to run a BBj application at install finish
INSTALL_BBEXEC_PROGRAM=$InstallDir/barista/sys/prog/bar_login.bbj
INSTALL_BBEXEC_CONFIG=$InstallDir/barista/sys/config/enu/barista.cfg
INSTALL_BBEXEC_WORKING_DIR=$InstallDir/barista
INSTALL_BBEXEC_TERMINAL=T0
INSTALL_BBEXEC_ARGS=aon^-uguest
INSTALL_BBEXEC_SYNC=false
INSTALL_BBEXEC_SHOW_PROGRESS=true
INSTALL_BBEXEC_ALLOW_CANCEL=false
INSTALL_BBEXEC_PROGRESS_TITLE=BASIS Custom Installer
INSTALL_BBEXEC_PROGRESS_TEXT=Finalizing AddonSoftware installation...
INSTALL_BBEXEC_FAILURE_TITLE=AddonSoftware Installation
INSTALL_BBEXEC_FAILURE_TEXT=AddonSoftware installation is complete.
INSTALL_BBEXEC_NUM_RETRIES=6
```

Figure 4. Defining a BBj program to run after installation completes

This ended up being a great win-win experience as “eating our own dog food” gave us the chance to “taste” some of the same BCI hurdles as our customers and implement solutions that benefited us in our immediate need, and BASIS customers down the road.

But Wait, There's More...

While our own marketing campaign was the catalyst for this string of improvements, shortly afterwards a customer came to us with some additional BCI requirements. They needed to 1) install their own product outside of the BASIS directory and 2) configure the uninstall wizard not to uninstall their product.

Seeing these customer requirements as viable enhancements, we added the ability to identify the installation drive via a new \$InstallDrive variable so that on Windows, for example, the developer could provide a full path in order to create shortcuts to their product offerings. In order to be exempt from the uninstall process, we added the **uninstall="never"** attribute on both the suite and feature nodes in the custominstall.xml file.

Summary

“So what,” you ask? Two things are worth noting. Firstly, the BCI is a very powerful and useful tool and it is included with BBj obviating the need to purchase a third party tool. If you are not already taking advantage of it, it might be time to exploit it. Secondly, and perhaps more importantly, if you are using the BCI and it currently doesn't have the features you want, don't be shy. Let us know what you need and if it makes sense, we will add it to this powerful building block! ■



Read *Custom Apps Install – Easy as 1- 2- 3!* at links.basis.com/11custominstall





Are You Prepared for Cloud Failure?

It's 3:00 A.M. and you awaken to the sound of a text on your phone. Is it the kids? Is it your mother-in-law? You focus on the bright screen in a pitch black room and discover it's neither the kids nor your mother-in-law. The server is down at work and it requires your immediate intervention. In a half-awake stupor, you drag yourself to your laptop and discover the server is not running at all. You think, "Wonderful, time to rush into work!" Just how long will you be down?

At BASIS, we run all of our servers – production, build, web – in the cloud, specifically the AWS (Amazon Web Services)



By Shaun Haney
Quality Assurance
Engineer

cloud. Among many other benefits, this prepares us for a quick recovery from text messages like this and allows us to finish a good night's sleep. This article reveals how we have prepared ourselves with the use of the latest technology from both the industry and BASIS.

What is AWS?

The AWS cloud is a collection of data centers that run virtual machines and charges for runtime, storage, and network usage. The cloud provides flexible configuration and backup possibilities that in-house LANs don't normally offer. When a cloud machine or "instance" goes down, it takes only a matter of seconds to create another identical instance from an "AMI" or Amazon machine image. The administrator saves permanent data on a virtual hard drive or "volume" and attaches it to an instance. The flexibility of virtual machines relieves system administrators from any concerns about hardware and allows them to focus on server and software configuration. As with the virtualization of so many other objects, physical limitations no longer apply: You can easily have 1 machine or 100 machines, all identical.

Amazon has several data centers around the world; in the US - California, Oregon, and Virginia; and worldwide - Brazil, Ireland, Japan, and Singapore, etc. If a server fails, one can usually solve the problem by connecting to a still-running instance and performing remote administration. It might be necessary to launch a new instance of the server if Amazon has experienced failures with its own (non-virtual) servers that caused an entire region or availability zone to become unavailable (see bit.ly/wXsGyz). Regions and availability zones becoming unavailable is problematic and extremely expensive for businesses.

Earlier this year on June 29th, the Washington D.C. Derecho storm (abcn.ws/PUBe2l) caused a power failure at one of Amazon's data centers (zd.net/KNQJRz). The power outage directly impacted companies like Netflix, Instagram, Pinterest, and Heroku. In particular, Netflix's streaming services were out for three hours at an extremely high peak usage time. During these failures, large service providers not only must recover from the outage, but also field a tremendous volume of customer service calls. In general, for companies that operate in the cloud but aren't prepared for regional outages, such an outage means the inability to conduct business for the duration of the outage and can also mean loss of data.

Be Prepared

So exactly what does it mean to "be prepared" for a cloud catastrophe? In short, being prepared means keeping redundant copies of data over multiple regions. BASIS employs three methods for backing up data and copying machines across several regions to maintain redundancy.

The first backup method BASIS uses is our own BBj® replication to copy our databases and files to machines in other regions. Configurable in Enterprise Manager, replication creates up-to-the minute copies of our data and BUI programs on several machines in other regions. Should we ever experience a failure in California, for example, we can recover from any one of several copies of our important files created at the time of the failure.

The second method backs up data using the Linux utility 'rsync' to copy the BASIS production system's drives each night to a failover copy of production. Our production machine is in a data center in one region separate from the region of the failover machine's data center. This means if we experienced such an outage, we could quickly launch the failover machine in the other region, which becomes our new production machine.

The third method BASIS uses is backups of data using a Linux backup utility called Duplicity. While the other two methods keep a single copy of data at a particular point in time, Duplicity keeps an incremental backup spanning a month's time. This way, if we find our up-to-the-minute and 24-hour-old data are faulty, we can recover from an earlier period of time when our data is still good. While data from the other two methods of backup is available instantly, data from

Duplicity is incremental so it requires a rebuild that could take a few hours.

In addition to maintaining copies of data, BASIS backs up all AMIs to other regions. Currently, Amazon does not provide a convenient way to back up machines across regions, so we have developed our own process. Basically, we copy the AMI's root volume to another region and then create a new AMI from that volume. Since AMIs change so seldom compared to live data, we only make backup copies of AMIs as we update them.

A Quick Recovery

BASIS employs these procedures daily in anticipation of a major outage. If or when this inevitable event might occur and our production server fails, we can switch (or "fail over") to another region with a quick recovery time of less than 5 minutes.

To execute this failover, the first step is to change the production server's DNS record so that it points to an IP address in the new region. Then we set our DNS up with a TTL of 300 seconds (5 minutes) and launch an instance of the AMI we copied over in the new region and attach the backup volume to the instance. Next, we rsync our replicated data to the backup volume and then assign the IP address to our instance that corresponds to the DNS record we just changed. Finally, we start up all the necessary services on our new instance and we're back in business. The entire process takes less than 5 minutes!

Conclusion

While running all BASIS' servers in the cloud allows us to continue business regardless of whether our employees are in the building, a cloud outage has the potential to bring business to a grinding halt. BASIS, however, incorporates geographical redundancy into its maintenance procedures as a strategy for quick recovery. This strategy ultimately allows BASIS to continue business as normal and serve you much more effectively than if we maintained all our servers on a single LAN within our building.

A cloud outage can result in immeasurable loss of money and data. Geographical redundancy is a vital strategy for quick recovery in the event of such a loss. Running all BASIS servers in the cloud allows us to continue regardless of whether our employees are onsite or virtual. This strategy ultimately allows BASIS or anyone to continue running business as normal and serve customers more effectively than from servers maintained on a single on-site LAN. Take it from us, setting up servers with redundancy in the cloud IS the best and most effective way to be prepared. If you are looking for a robust, affordable ERP cloud solution, you would be hard pressed to find one that is more architecturally reliable and redundant than the AddonSoftware® Cloud solution, because it comes with this BASIS tried-and-tested robust redundancy and recovery system, right out of the box! ■



Read *BASIS Survived Amazon Outage* at links.basis.com/12survived



Java Breaks Deliver Solving the Information Delivery Challenge

A few years ago, we realized BASIS was facing an information delivery problem - too much content to deliver in a timely manner, to a "too busy" audience located around the globe.

While we were successful with TechCons and TechViews, they really only reached a very narrow sector of our community. TechCon conferences continue to present great opportunities to speak face-to-face with our partners about the ever-expanding BASIS technology, features, and functions built into the product set, but are only held every 18-24 months. How do we reach our community with more current and frequent information? BASIS TechViews took mini sessions to cities around the U.S. but were difficult to target geographically to maximize and justify one-day attendance, and were expensive for us to conduct with any frequency. How do we host regular 'bite-sized' presentations that wouldn't monopolize too much time so that our community could attend while at work, at home or in a hotel; from any city across the country or around the world?

The solution to this dilemma came in the summer of 2009. We decided to jump on the webinar bandwagon and host sessions on current topics for about 30-minutes during which our attendees might enjoy a cup of coffee...a virtual "coffee break" of sorts. We tossed around several names but "Java Break," with our Java-infused BBJ® product as the headliner, seemed to be the obvious choice. Committing to host or "serve up" these breaks bi-weekly, we designed them strategically to educate and inform resellers and end-user developers alike using demonstrations and presentations of the latest BASIS technology. We settled on the recurring start time of 10 AM Mountain Time to conform with the typical work day in the continental U.S. and also to be reasonably convenient to our customers who span at least 10 time zones.



By Paul D. Yeomans
Vertical Market
Account Manager

In the fall of 2009 at TechCon in Albuquerque, we announced the Java Break concept and logo, printed on ceramic coffee cups, and distributed them with a schedule of the first Java Break series.

Now, three years later, we have registered a total of nearly 3,200 attendees for more than 60 sessions. Combining input from our developers and end users, and comments in the discussion forums with new product features and industry trends, we deliver a wide range of topics to an audience that now has the time, interest, and opportunity to attend. The result – Java Break topics that cover the practical applications of BASIS technology including language/interpreter, development tools, database management, system administration, utilities, and the application building blocks of the AddonSoftware® by Barista® ERP solution. In our most popular sessions we highlighted enabling data replication and auditing for your PRO/5®, Visual PRO/5®, and BBJ apps; adding browser access to PRO/5 data, and imparting email functionality to your PRO/5 apps. Many organizations use the BASIS Java Break as semi-monthly opportunities to bring their development team together for collaboration and brainstorming while viewing the session.

Looking back at the goals of our solution, we've succeeded...

...bite-sized presentations

Generally no more than 30 minutes

...accessible from work, home, or hotel

Web-based sessions only require an Internet connection

...from any city

Global coverage with attendees from Argentina, Australia, Bangladesh, Bulgaria, Canada, Chile, Colombia, Dominican Republic, England, Germany, Guatemala, India, Italy, Macedonia, Mexico, Netherlands, Nigeria, Poland, Puerto Rico, Serbia, Sweden, Switzerland...

For some it is an evening's diversion, though for our Australian friends, it is 'bloody early.'

...on your choice of viewing device

Viewable from your desktop, laptop, mobile device, or from your large screen TV using Airplay for iPhone/iPad

...on demand

As the Java Breaks have evolved, so have their longevity, adding **anytime ease** -

All Java Breaks are available to the community *anytime* from either basis.com/java-break-basis or the [BASIS YouTube](#) channel. In fact, the YouTube channel alone has nearly 10,000 video views with dozens of subscribers, and counting.

A most unanticipated and invaluable resource from these Java Breaks is the live question and answer sessions that we record, transcribe, and post as a Google Doc to the archive page (basis.com/java-break-basis). These Q&A sessions give us real-time input and interaction with our community to help guide the direction of future Java Breaks and potentially, our products. Also, posting the Q&As in Google enables us to easily edit and update as we enhance our technology.

Have Java Breaks been worthwhile to the BASIS community from the customer's perspective?

Here are just a few comments we received:

"It was great to see how easy it is to serve and consume Web services. Thanks for these Java Breaks. They are wonderful."

"Preserving Modifications Through Upgrades was very well done and answered several questions I had. It sparked new ideas of what I can do with Barista"

"Thank you for providing the time and facilities to get this great information out."

"This Java Break certainly stirred up a nest of worms that are already swimming around in my head. For that, I am grateful!!"

"Writing Your Web App in the Latest BBx - The CSS was exciting and very enjoyable. I can't wait to get started."

Thank you for 'taking a Java Break' or two with us. Together, we've clearly brewed a smashing success! Information delivery challenge **SOLVED.** ■



If you have not yet joined us for a live Java Break, check out our schedule at links.basis.com/events and register there or from our Monday morning email announcement. In the meantime, catch up on past Java Breaks at links.basis.com/javabreak!

Drowning in unfinished projects?

Don't off-shore, near-shore, or out-source...
Smart-source with



- Develop, maintain, and migrate all Business BASIC-based applications
- Outsource IT services permanently
- Support all BBx[®] generations – PRO/5[®], Visual PRO/5[®], BBj[®], and Barista[®]
 - Managed in English through BASIS US and Europe
 - Committed to
 - Globally unified quality standards and project management guidelines
 - Legal peace-of-mind, thanks to US and EU contract standards

Smart-source services provide both ISVs and corporate IT departments with trained, certified, and qualified software architects and developers staffed through the global network of BASIS partners and subsidiaries.

US - BASIS International Ltd. info@basis.com

Europe - BASIS Europe Distribution GmbH eu@basis.com



BBj Documentation is as Easy as JavaDocs

W

Wikipedia defines [Javadoc](#) as “a documentation generator from Sun Microsystems [now Oracle] for generating API documentation in HTML format from Java source code. The HTML format is used to add the convenience of being able to hyperlink related documents together. The ‘doc comments’ format used by Javadoc is the de facto industry standard for documenting Java classes.”

With the help of a new BASIS utility, [BBjToJavadoc](#), and a little effort on your part, you now have a second motivation to put comments in your code, because BBjToJavadoc also gives you the ability to rapidly create API documentation from your BBj® application code using this new BASIS-supplied Javadoc documentation generator.

Figure 1 shows an example of doc comments in BBj code. In this example, the “block tags” are `@param`, `@return`, and `@see`.

```
rem /**
rem * Method doDDX:
rem * Exchange string data between dialog control and program variable
rem * @param BBjString Constructed control Name
rem * @param BBjString Program variable contents
rem * @param BBjNumber Init Flag: 0=Get control data, 1=Set control data
rem * @param BBjNumber Required Flag: Check for empty content when getting data from control
rem * @return BBjString Control data
rem * @see getControlByName()
rem */
method protected BBjString doDDX(BBjString pCtrlName$, BBjString pCtrlContents$, BBjNumber pInitFlag, BBjNumber pRequired)
```

Figure 1. BBj code incorporating doc comments



By Ralph Lance
Software Engineer

The resulting HTML generated would then look like **Figure 2**.

doDDX

```
protected BBJString doDDX(BBJString pCtrlName$,
    BBJString pCtrlContents$,
    BBJNumber pInitFlag,
    BBJNumber pRequired)
```

Method doDDX: Exchange string data between dialog control and program variable

Parameters:

BBJString - Constructed control Name
 BBJString - Program variable contents
 BBJNumber - Init Flag: 0 = Get control data, 1 = Set control data
 BBJNumber - Required Flag: Check for empty content when getting data from control

Returns:

BBJString Control data

See Also:

getControlByName()

Figure 2. Resultant HTML API documentation example

With BBJToJavadoc, BASIS has made it possible for you to embed similar documentation comments in your BBJ programs and generate your own documentation in HTML Javadoc format. In particular, your BBJ custom classes can take advantage of some of the more advanced documentation capabilities, allowing you to navigate easily via hyperlinks among your classes. Because the Javadoc engine from the Java JDK toolset is used in the process, any valid document tag can be embedded in the doc comments.

Figure 3 is an example of a BBJ custom dialog class with some doc comments that the Dialog Wizard embedded for us. Follow the required few simple steps below to generate the corresponding Javadoc documentation.

```
field public BBJTopLevelWindow Wnd!

rem /**
rem * Constructor Custmaintbig
rem */
method public Custmaintbig()
    #super!("custmaintbig.arc",100)
    if stbl("+USER_LOCALE",err=*endif) <> "" then
        #ClientLocale$ = stbl!("+USER_LOCALE")
        #Translator! = BBTranslator.getInstance("Custmaintbig",#ClientLocale$, "en",#PgmDirectory$)
    endif
    #Wnd! = #super!.getWndTop()
    DialogUtils.buildDialogProperties(#Translator!,#super!.getCtrlVect())
    if #Wnd! <> null() then
        #initToolBar()
        #setCallbacks()
    endif
methodend

rem /**
rem * Method initToolBar:
rem * Setup toolbar
rem */
method private void initToolBar()
rem /** DLGWIZ_BAR_BEGIN */
rem /** DLGWIZ_BAR_END */
methodend

rem /**
rem * Method setCallbacks:
rem * Set control callbacks
rem */
method private void setCallbacks()
rem /** DLGWIZ_CBS_BEGIN */
rem /** DLGWIZ_CBS_END */
methodend

    #super!.getControlByName("CustMaintWindow").setCallback(#API!.ON_CLOSE,#this!,"CustMaintWindow__ON_CLOSE")
rem /** DLGWIZ_CBS_END */
methodend
```

Figure 3. BBJ custom dialog class with embedded doc comments

Step 1. Start the Wizard

To start the wizard, run the program called **BBjToJavadocWizardRun.bb** located in the **<BBjHome>/utils/BBjToJavadoc** folder.

Once the wizard launches (**Figure 4**), stipulate where to store the generated documentation and choose the level of scope, or visibility, the documentation includes - public, protected, or private. Choosing a 'private' level of scope documents everything.

Lastly, choose one or more non-tokenized BBj source files to document.

Step 2. Set Options

After clicking the [Next] button, proceed to the second screen (**Figure 5**) to set options for the Javadoc program. Read more about these options in the online [Javadoc documentation](#).

Step 3. Generate the Docs

Now, you are ready to generate the actual documentation, so click [Next] and then [Finish] to display the completed screen shown in (**Figure 6**). You have the option to show the results of the generation immediately upon completion.

If desired, the Wizard displays the nicely structured HTML documentation in your default browser (**Figure 7**). You can easily navigate between the different regions of the document - the field summary, constructors, and the methods themselves. Clicking the hyperlinks for methods or fields in the left navigation pane takes you to a copy of the code that defines the entity.

So just that easily, the BBjToJavadoc utility has done all the work for you, simply creating a robust hyperlinked documentation system derived from your own code.

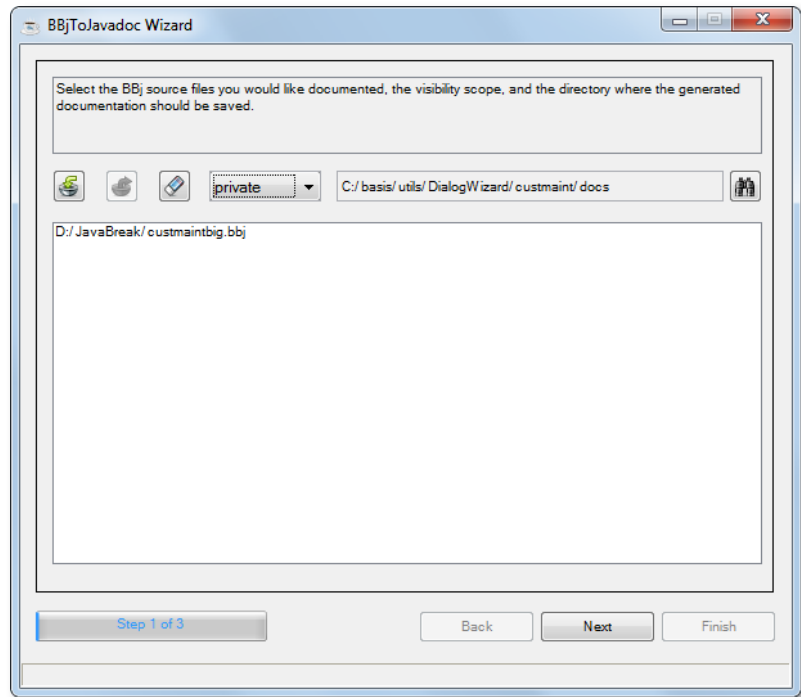


Figure 4. Wizard screen for Step 1

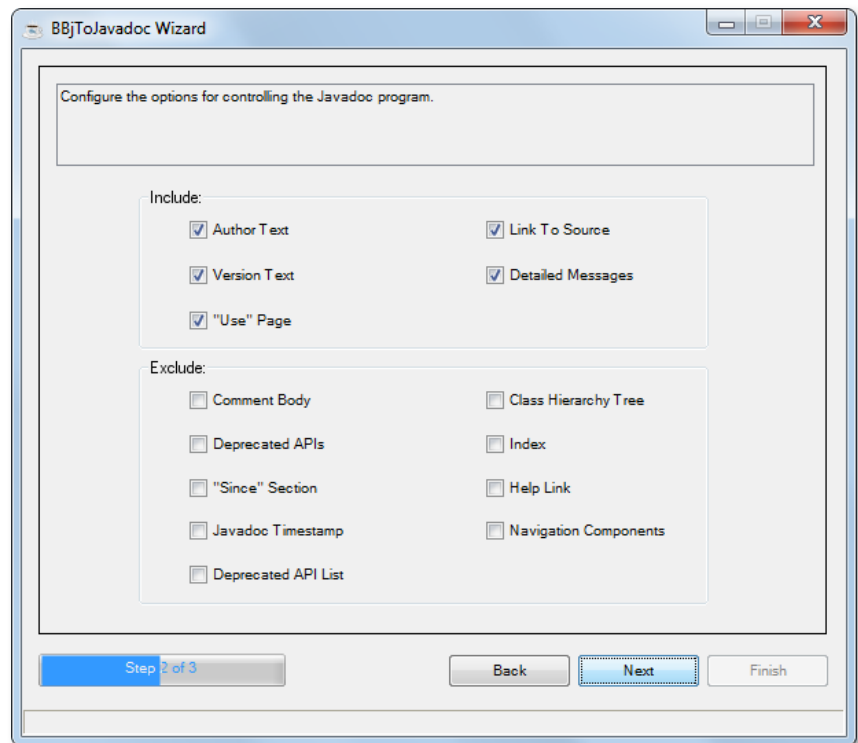


Figure 5. Wizard screen for Step 2

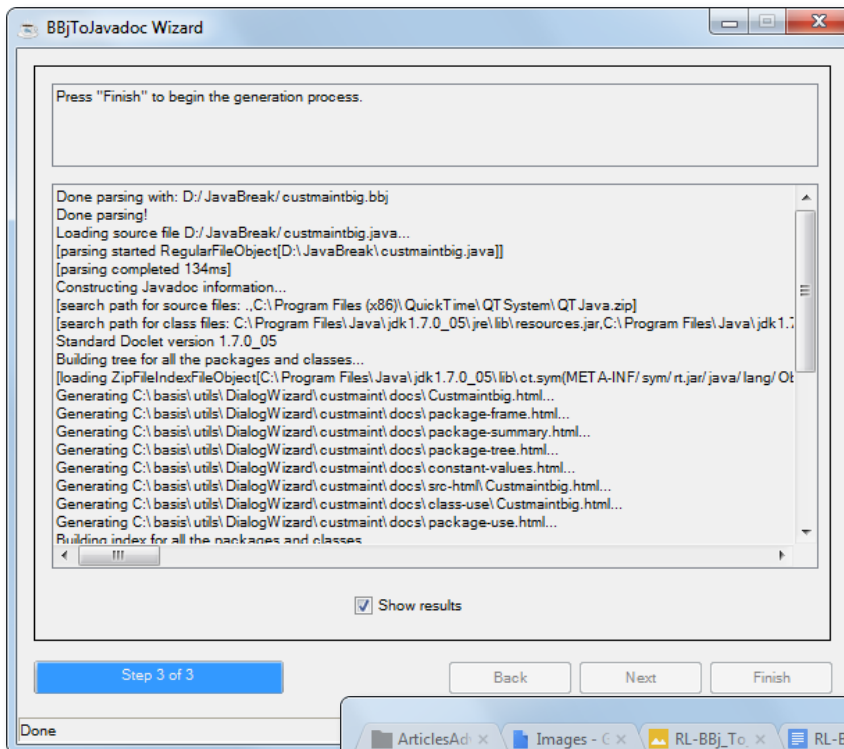


Figure 6. Wizard screen for Step 3

Summary

BBjToJavadoc is a documentation generator from BASIS for generating API documentation in HTML format from BBj source code. The HTML format is used to add the convenience of being able to hyperlink related documents together. The 'doc comments' format used by BBjToJavadoc is the BASIS standard for documenting BBj custom classes. While we can't actually remove the often neglected task of documenting your code from your 'to-do' list, this BASIS building block utility makes it as easy as possible for you to write self-documenting object-oriented code. ■

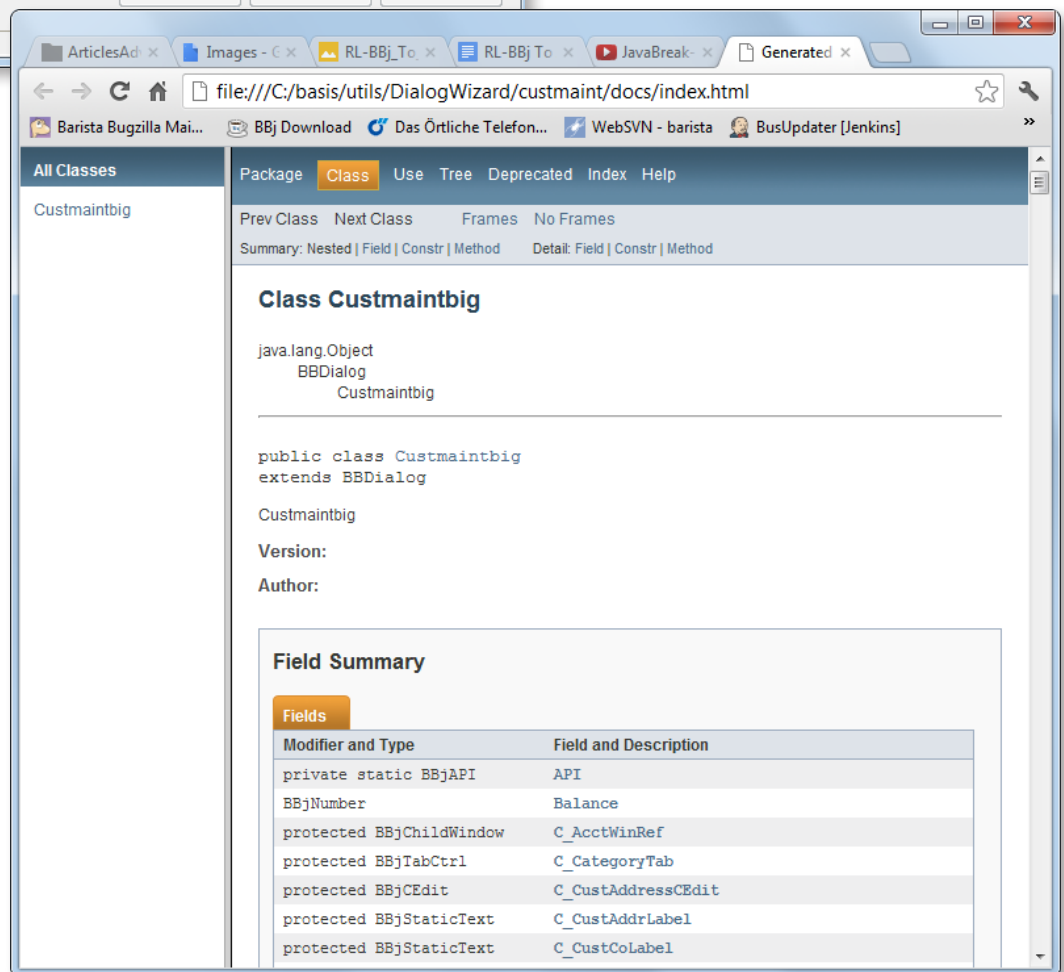


Figure 7. Resultant HTML documentation



For more information, visit

- BBjToJavadoc - links.basis.com/bbjtojavadoc
- Oracle Java API Documentation Generator - bit.ly/TFz1mJ
- Oracle Javadoc Tool - bit.ly/9I0Dd6

Any system purchased 'off-the-shelf' today is powerful enough to run BBj® and its features. That dusty Windows XP in the corner of your office likely has enough horsepower to run any thin client application developed in BBj. It is probably even powerful enough to develop apps in the BASIS IDE or the Barista® Application Framework. But ask our sales or support teams what the system requirements are for a full application deployment and they will tell you the same thing: "*It depends.*"

The fact is, there are simply too many variables involved in a BBj application deployment for our sales or support teams to provide such specific information. This is true of any enterprise level Java- or .NET-based application: Only the application developer and end user truly know the system requirements of a deployment.

The good news is that all the tools necessary to assess the requirements for your deployment are at your fingertips.

Q. What factors should I consider in a BBj deployment?

A. You need to consider these important hardware and software factors:

Topology – The ability to distribute BBj database management, interpreter, and client functionality across multiple systems (or 'tiers') means vast varieties of application topologies. You will need to consider the system requirements for all of the tiers involved (see **Figure 1**).

Will your deployment be a **single-tier** CUI application on a UNIX server with terminal emulator clients or will it be a **two-tier** GUI deployment on a Windows Terminal Server farm with RDP clients? The former single-tier UNIX deployment actually requires less memory than a similar PRO/5® deployment, averaging only 2 MB for each additional user. The latter 2-tier Terminal Server deployment could require considerable system resources, depending upon the memory requirements of the application itself and the number of concurrent users. **Three-tier** deployments, where the 'Presentation Layer,' 'Application Layer,' and 'Data Layer' all reside on separate systems – clients, servers, or clusters of servers can have widely-varying memory, CPU, and disk I/O requirements at each layer. Disk I/O will be more of a factor on a Database Management Server (DBMS) than an Application server; an Application server will have greater memory requirements than a DBMS server.

For an in-depth discussion of different BBj deployment strategies, see *Choices, Choices, Choices* at links.basis.com/05choices.

Application – No two applications use the same amount of resources. A CUI data-entry program would have very low memory overhead. Does your GUI application have a lot of forms or controls? If so, memory requirements on the client side could be a factor.

Number of concurrent users – How many users do you anticipate will log in to your application in the peak hours on a given site? Again, if your deployment is a single-tier CUI application, each additional user will have a minimal impact on system resources.

However, a Terminal Services deployment will be a bit more complex. Here, the 'N+1 Rule' is in effect, where the number of JVMs invoked on the system will equal the number of logged-in users plus one extra JVM for BBj Services powering the DBMS.

In Terminal Server/Citrix deployments, there is greater overhead involved in the first invocation of the JVM housing the DBMS, but marginally less memory used for each subsequent invocation. One can only deduce a true picture of memory consumption after adding five or more users. There is no sharing possible in the application server JVM because each user/display must run in its own JVM. Of course, it is important to remember that the maximum memory used by all concurrent users plus BBj Services should not exceed the physical memory of the system.

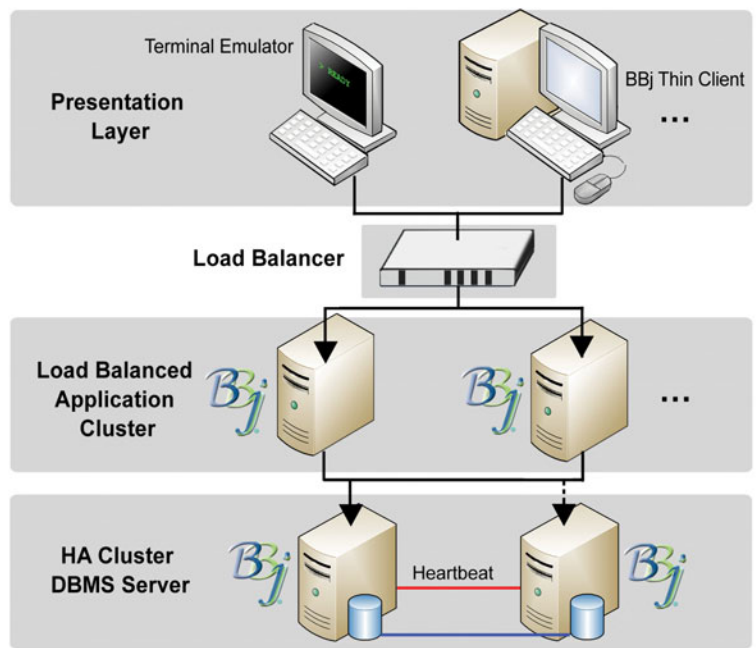


Figure 1. Typical 3-tier BBj deployment in which layer interaction occurs over the network



By Bruce Gardner
Technical Support
Supervisor

Virtualization – While it has the potential to reduce cost, you must handle virtualized deployments carefully. Multiple guests competing for resources on a single host server will have a negative impact on disk I/O and introduce network latency.

Network latency – As discussed in the **Topology** paragraph, multi-tier application deployments can place very different demands on systems in the separate layers, depending on their function. However, the distribution of presentation, application, and data functions to different systems necessarily means that the interaction between the layers will be 'over the wire.' Network latency can be the most important factor in multi-tier deployments, and you should pay attention to the latency effects between the layers.

Q. How do I determine the system specifications for my deployment?

A. Match your Test environment to the target Production environment and run it through its paces.

In an ideal world, all testing would occur in a test environment that exactly matches the target production environment. While some developers have done a spectacular job of this, it can be an unrealistic and impractical goal for most. Where it is not possible to match a test environment to a production environment, BBj, system, and third party tools can be used in the test environment to extrapolate a relative and predictive baseline. The closer you match the target production environment operating systems, topology, and workload, the more accurate your assessment. Use a meaningful approximation of the application workload. A tight loop that executes thousands of reads and writes in a minute does NOT approximate user interactive workload!

You will find that the data you collect in this process will inform your n-tier deployment decisions. For example: An interactive data entry application will not be sensitive to network latency in 3-tier deployments, but it might make sense to combine the 'application layer' and 'data layer' into one layer for batch jobs and reports, taking network latency between the application and DBMS servers out of the picture.

Q. What tools should I use?

A. The more, the better! Here are a few examples:

- **BBj tools** – BBj's Enterprise Manager can provide a great deal of information about the server deployment.
 - **Memory Usage module** – Provides real-time graph of current memory usage. You may also view your customer's memory usage simply by pointing the module to a copy of their BBj logs.
 - **System Logs module** – Provides detailed, minute-by-minute, information on current memory usage and Garbage Collection durations. This information can help 'tune' memory-related Java arguments for BBj Services.
 - **BBj Processes module** – Provides a detailed view of the BBj processes currently running.
 - **BBj File System module** – Provides a detailed view of all files currently open on the server.
- **System tools** – Know your target operating system and the tools that come with it. Utilities, such as 'top' on UNIX and 'Task Manager' on Windows, can provide a great deal of information about CPU, memory usage, and disk I/O. Once your test environment is in-place, incrementally add users and watch the resource utilization on the server. After a number of users have been added with a simulated workload, you should be able to extrapolate how much CPU, memory, disk I/O, etc. will be utilized during peak production hours.
- **Third party tools** – Third party tools, such as Microsoft's free [Sysinternals](#), can provide detailed information for just about any metric you care to measure on a Windows system. Other third party tools can help simulate a production environment by adding a non-BBj workload or simulating network latency.
- **Test programs** – It is not always possible to approximate a realistic workload on a server but you can always write programs that will: Simulate an interactive workload by spawning BBj processes, or write a program that performs thousands of file operations to simulate a batch process.

Q. Should I re-evaluate system requirements as newer versions of BBj and Java are released?

A. Yes. Java and BBj are constantly changing. New Java/BBj features might require more or less resources. Optimizations are always forthcoming as well; Java introduced dynamic memory allocation in Java 1.6.0_18, taking much of the guesswork out of memory-related Java arguments for BBj Services. Recent filesystem refactoring in BBj 12.10 have resulted in much faster concurrent file access.

Summary

There is no substitute for a real-world test that simulates the actual behavior of your BBj deployment. Only you, the developer, can truly determine the specific needs of your application. A full understanding of the dynamic factors involved in all deployments, coupled with accurate testing and use of tools, will go a long way towards determining those requirements. ■



- Read the BASIS Advantage article *Choices, Choices, Choices* at links.basis.com/05choices
- Check out Microsoft's free third party Sysinternals tools at links.basis.com/iydfu



Use our **ERP modules** as the core building blocks for your vertical solution. We take care of the common ERP elements while you focus on your areas of expertise.

Our **commercial open source** ERP model is a truly revolutionary partnership that blends the best of the open source approach with meaningful incentives to build your own profitable business.

Learn more about partner opportunities at info@addonsoftware.com



AddonSoftware by BASIS International Ltd. – the big little software company

