Blowing the Doors Wide Open With ClientObjects

By David Wallwork

BBj® introduces enhancements and features with each new release. One of these features, embedded Java code, allows the developer to embed Java code within their BBj programs in order to take advantage of Java's advanced data types, JVM-specific information, and the huge body of readily-available Java code and libraries. In BBj 8.0, the concept of embedding Java code is extended through the introduction of ClientObjects. This article explores how developers can use the new ClientObject feature to create and implement client-side Java Objects including third party classes as well as custom classes; GUI controls as well as non-GUI Objects.

The Need for Client Objects

Consider a sample program (see **Figure 1**) that reads key-value pairs from a channel and places them into a Java HashMap.

```
1 REM read all key and values from file and place them into a HashMap
2 map! = new java.util.HashMap()
3 while 1
4         key$=key(chan,end=*break)
5         read record(chan)value$
6         map!.put(key$,value$)
7 wend
```

Figure 1. Read and place key-value pairs into a Java HashMap

This example creates a HashMap within the JVM of the interpreter, which is often what developers want. However, there are times when they need to create a Java Object within the JVM of the client rather than within the JVM of the server. For instance, to print the current time, one might write the code shown in **Figure 2**.

```
1 REM print the date/time on the server machine
2 now! = new java.util.Date()
3 print now!
```

Figure 2. Print the current date and time for the server, rather than the client

This is all well and good if the server and the client are in the same time zone, but what happens when the server and the client are in different time zones? BBj 8.0's ClientObjects come to the rescue. To print the date and time of the client, the program must use ClientObjects. ClientObjects use an @ symbol to indicate that a Java Object should be created within the *client* JVM rather than within the *server* JVM. Simply adding the @ symbol to our example in **Figure 3** creates a Date Object on the client machine and prints the date and time of the client.

```
1 REM print the date/time on the client machine
2 now! = new java.util.Date@()
3 print now!
```

Figure 3. Add an @ to print the date/time on the client

Background and Terminology

Because the interaction between the BBj server and the BBj client is seamless, most developers do not think about the difference between the server-side environment and the client-side environment. In order to understand ClientObjects, it is important to understand this distinction.

In most common deployments, the client and the server are on different machines, so it is easy to distinguish between them. Even when the client and server are on the same machine; even if they are running within the same JVM, we still make a distinction. We refer to any object created within the JVM of BBjServices as a server-side object. Any object created within the JVM of the client is a client-side object. For clarity of discussion, assume that the client and server are running on separate machines.

Under the covers, ClientObjects are simply Remote Method Invocation (RMI) handles to the client-side Java Object. When executing the code in **Figure 3**, line 2 creates two Java objects; one is a client-side instance of java.util.Date, the other is a ClientObject that is an RMI handle to the client-side instance of java.util.Date.

The ClientObject has the same public methods as the client-side object so the BBj program can call the same methods on `now!` in **Figure 3** as it can call on `now!` in **Figure 2**. Though both variables look the same to the program, the variable in **Figure 2** holds a server-side object while the variable in Example 3 holds an RMI handle to a client-side object.

continued...

David Wallwork
Senior Software Engineer

This distinction – whether an Object is in the client JVM or in the server JVM – is especially important when using GUI Objects. A GUI Object that is created in the server JVM will display on the server console. A GUI Object that is created in the client JVM will display on the client console.

Seeing Client Objects is Believing

BASIS created several demo programs to illustrate the myriad of features introduced in BBj 8.0. Many of these demos use ClientObjects to perform in ways that were not possible prior to 8.0.

First, run the "LaunchDock" demo† that is based on ClientObjects. Notice the floating tool button that displays menu selections as you mouse over the buttons. Next, launch the "Web Browser" demo† to see a fully-featured Web browser controlled by a BBj program. As with all the demos, the full code is available for your review in its appropriately named folder.

Here are several noteworthy aspects about the "Embedded Web Browser" demo program:

- The actual GUI component seen on the client screen is an instance of `org.jdesktop.jdic.browser.WebBrowser`, which is an open-source Java class downloadable free of charge from www.jdic.dev.java.net.

  Using this class points out a powerful benefit of ClientObjects – developers can now instantly create applications with controls that are not part of the BBj language. Instead of requesting that BASIS write a Web browser control from scratch for BBj, developers can download a Java-based Web browser from the Internet and begin using it right away.

- The BBjWrappedJComponent wraps the WebBrowser object resulting in a BBjControl with a control ID and all the methods defined for a BBjControl

- The BBj program responds to Java events generated by the WebBrowser object

The "Embedded Web Browser" demo† is a fully developed application and at first glance may seem intimidating to the reader who is not familiar with CustomObjects and ClientObjects. However, the following sections provide simpler examples of the basic concepts used in this demo.

Displaying Java GUI Components on the Client

Since the introduction of embedded Java (server-side objects), a BBj program has been able to display Java GUI components as illustrated in **Figure 4.**

```
1  use javax.swing.JFrame
2  use javax.swing.JButton
3
4  frame! = new JFrame("Server Side")
5  frame!.setBounds(200,200,400,300)
6  pane! = frame!.getContentPane()
7  pane!.setLayout(null())
8
9  button! = new JButton("Button")
10 button!.setBounds(50,50,200,100)
11 pane!.add(button!)
12
13 frame!.setVisible(1)
14
15 escape
```

Figure 4. Display the Java GUI component on the server

```
1  use javax.swing.JFrame
2  use javax.swing.JButton
3
4  frame! = new JFrame@("Client Side")
5  frame!.setBounds(200,200,400,300)
6  pane! = frame!.getContentPane()
7  pane!.setLayout(null())
8
9  button! = new JButton@("Button")
10 button!.setBounds(50,50,200,100)
11 pane!.add(button!)
12
13 frame!.setVisible(1)
14
15 escape
```

Figure 5. Display the Java GUI component on the client

The problem with the sample in **Figure 4,** however, is that the JFrame and JButton display on the server, which is not of much use when the client is on a different machine. Simply adding an @ symbol as shown in **Figure 5** changes the server-side objects into ClientObjects so that these entities display on the client machine rather than on the server machine.

continued...

Introducing BBjWrappedJComponent and Adding a Java GUI Component to a BBjWindow

BBj introduced a new BBjControl named BBjWrappedJComponent in version 8.0 so developers can 'wrap' a JComponent and place it onto a BBjWindow. The program in **Figure 6** creates a BBjWindow and adds a BBjButton to the window. It then creates a JSplitPane, wraps the JSplitPane in a BBjWrappedJComponent, and places the component onto the BBj Window.

```
1  use javax.swing.JSplitPane
2  use javax.swing.JButton
3  use java.awt.Dimension
4
5  REM create a BBjWindow
6  open (unt)"X0"
7  sysgui!=bbjapi().getSysGui()
8  window! = sysgui!.addWindow(10,10,500,400,"BBjWindow",$00010003$)
9  button! = window!.addButton(1111,20,20,80,60,"BBjButton")
10
11 REM create a JSplitPane and place it on the BBjWindow
12 splitPane! = new JSplitPane@()
13 wrapped! = window!.addWrappedJComponent(2222,0,200,500,200, splitPane!)
14
15 REM add something to each side of the JSplitPane
16 splitPane!.setContinuousLayout(1)
17 splitPane!.setOrientation(javax.swing.JSplitPane.HORIZONTAL_SPLIT)
18 leftButton! = new JButton@("Left Java Button")
19 rightButton! = new JButton@("Right Java Button")
20 splitPane!.add(leftButton!, javax.swing.JSplitPane.LEFT)
21 splitPane!.add(rightButton!, javax.swing.JSplitPane.RIGHT)
22 splitPane!.validate()
23
24 process_events
```

Figure 6. Wrap the JSplitPane in a BBjWrappedJComponent

The `wrapped!` variable on line 13 is a BBj component and has a control ID. A developer can manipulate it using the methods available on any BBj component while still being able to manipulate the 'wrapped' JSplitPane with all the public methods of a JSplitPane.

Adding a BBjControl to a Java Component

BBj allows a BBjControl to be passed to the add method of a JComponent. The example in **Figure 7** removes the BBjControl from the BBjWindow and places it directly onto a JPanel that appears on the left side of the JSlider.

```
1  use javax.swing.JSplitPane
2  use javax.swing.JButton
3  use java.awt.Dimension
4  use java.awt.GridLayout
5  use javax.swing.JPanel
6
7  REM create a BBjWindow
8  open (unt)"X0"
9  sysgui!=bbjapi().getSysGui()
10 window! = sysgui!.addWindow(10,10,500,400,"BBjWindow",$00010003$)
11 button! = window!.addButton(1111,20,20,80,60,"BBjButton")
12
13 REM create a JSplitPane and place it on the BBjWindow
14 splitPane! = new JSplitPane@()
15 window!.addWrappedJComponent(2222,0,200,500,200, splitPane!)
16
17 REM add something to each side of the JSplitPane
18 splitPane!.setContinuousLayout(1)
19 splitPane!.setOrientation(javax.swing.JSplitPane.HORIZONTAL_SPLIT)
20
21 leftButton! = new JButton@("Left Java Button")
22 rightButton! = new JButton@("Right Java Button")
23
24 layout! = new GridLayout@(6,1)
25 leftPanel! = new JPanel@(layout!)
26 leftPanel!.add(button!)
27 leftPanel!.add(leftButton!)
28
29 splitPane!.add(leftPanel!, javax.swing.JSplitPane.LEFT)
30 splitPane!.add(rightButton!, javax.swing.JSplitPane.RIGHT)
31 splitPane!.validate()
32
33 process_events
```

Figure 7. Place the BBjControl on a JPanel (line 26)

continued...

Run the code `ClientObj-Fig7.src` shown in **Figure 7** and adjust the splitter of the JSplitPane. Notice the BBjButton changes size, as illustrated in **Figure 8**.

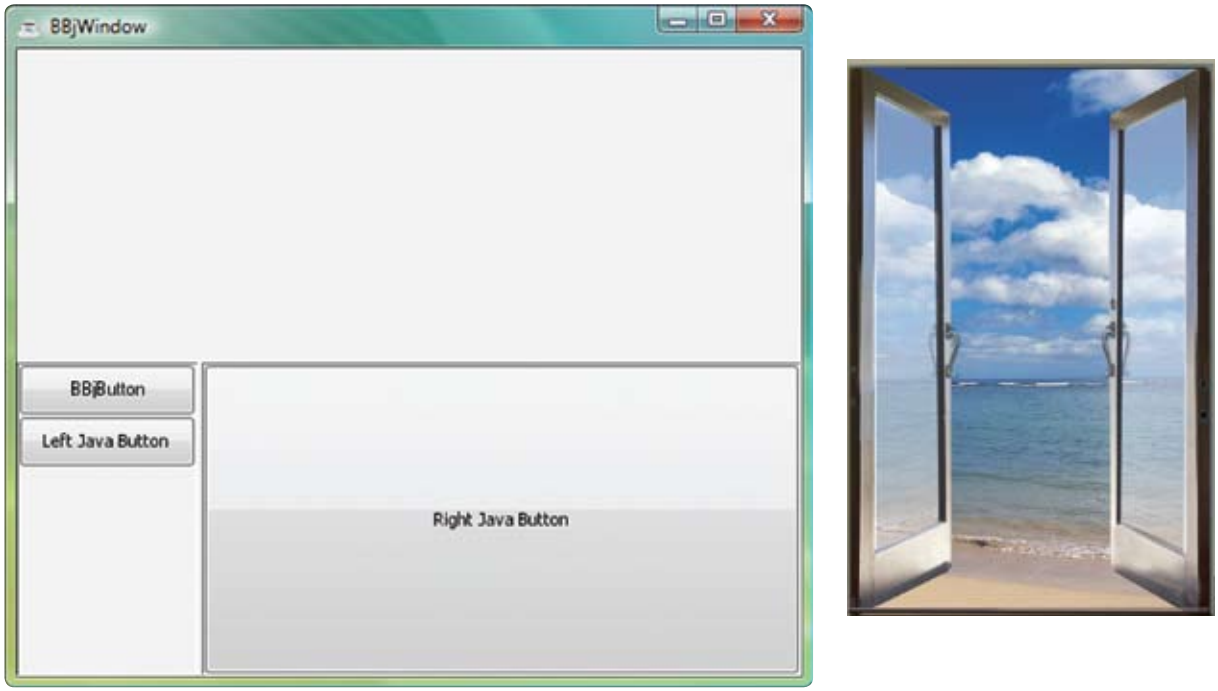

Figure 8. The result of running the code in Figure 7

Responding to Java Events Within a BBj Program

Most Java GUI components generate events in response to user actions. A BBj program can receive events that client-side Objects generate and the program can respond to those events using BBj code. The BBj code can manipulate the GUI controls or it may respond by only executing more traditional BBj code. In order to respond to events, a BBj program must use CustomObjects.

To see how to place a JSlider onto a BBjWindow and listen for the events generated by the JSlider, download the samples listed at the end of this article and run `ClientObj-Fig9.src`. Running this example prints the ChangeEvent and the value of the slider each time the user moves the slider.

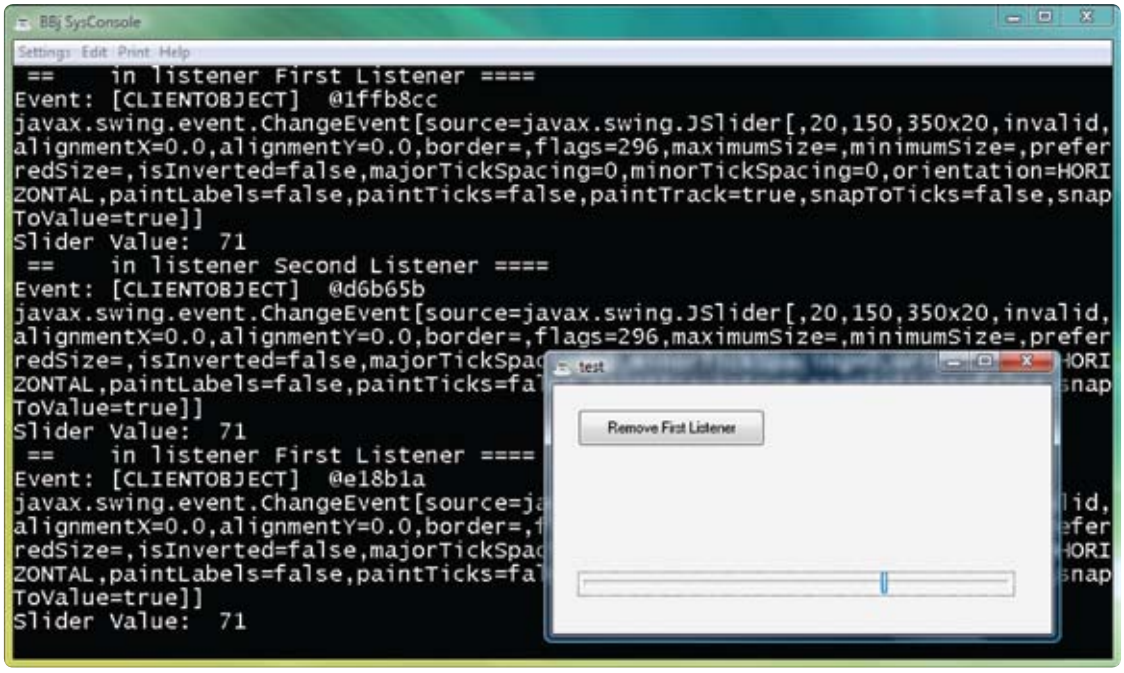

Figure 9. Print the ChangeEvent and the value of the slider

continued...

Realizing the Benefit of ClientObjects

ClientObjects allow the BBj developer to take advantage of Java components that the open-source community and in-house Java developers have written. The integration of these third party controls into a BBj application is a seamless extension of the more traditional use of embedded Java. The RMI functionality of ClientObjects allows manipulation of the third party control in essentially the same way as within a native Java application. This integration allows the developer to mix-and-match the BBj controls, data structures, and other business-oriented features of BBj seamlessly with Java GUI controls as well as other non-GUI Java classes.

Registering ClientObject Jars for Deployment

BBj expects to find the Java class files for client-side objects in jar files that either are jar files in the JDK or are registered jar files. If a ClientObject references a class that is not part of the JDK, then BBj will check the jar file that contains the class to see if the jar file is a registered jar. If the jar is not registered, then BBj will display a nag message to the user asking to place the class into a registered jar.

There are two ways to register a jar; by using the executable jar BASIS provides or by executing a Java command.

To use the executable jar, enter the following command at the command prompt after setting the appropriate classpath:

```
java com.basis.jarreg.client.JarRegistrar inputJarName outputJarName
```

In order to register a jar from within BBj or from within a Java program, execute the following command

```
com.basis.jarreg.client.JarRegistrar.registerJar (inputJarName, outputJarName)
```

These commands create a registered copy of the input jar file. The newly created copy contains additional information in the `META-INF/BASIS.KEY` file that BBj uses in order to recognize that this new jar file is a registered jar. If anyone modifies the content of a registered jar, the registration becomes invalid and the jar must be re-registered.

When a jar is registered, the JarRegistrar first establishes an internet connection to BASIS to determine the current version of BBj that is available from BASIS. Next, it places that version number in the file META-INF/BASIS.KEY within the output jar. Registered jars are forward compatible; registered for all present and future BBj versions. This means, for example, that a developer registers a jar during 2009 when the current license version is 9.xx, then that jar is recognized as a registered jar so long as BBjServices runs with a 9.0 license or greater. If BBjServices uses an 8.0 license, then it will display a nag message.

Developers do not need to use registered jars during development if they are running with a DVK (Developer's Version Kit) license. When running with a DVK license, available from BASIS Sales, BBj uses any Java class as a ClientObject without generating a nag message.

Testing and Deploying a Registered Jar File

When developing an application in an environment with a DVK license, it is very important to test the application with the DVK feature disabled prior to deploying the application. To run BBj with the DVK license disabled, follow theses steps.

1. Log in to BBj Services using the Enterprise Manager.
2. Select the **Server Information** item from the navigation list at the left side of the Enterprise Manager.
3. Click on the **Performance** tab in the right side informational area.
4. Choose the **Production** radio button for the **Use DVK License If Present** option shown in **Figure 10**.
5. Click on the [Save] button to save your new settings.

continued...

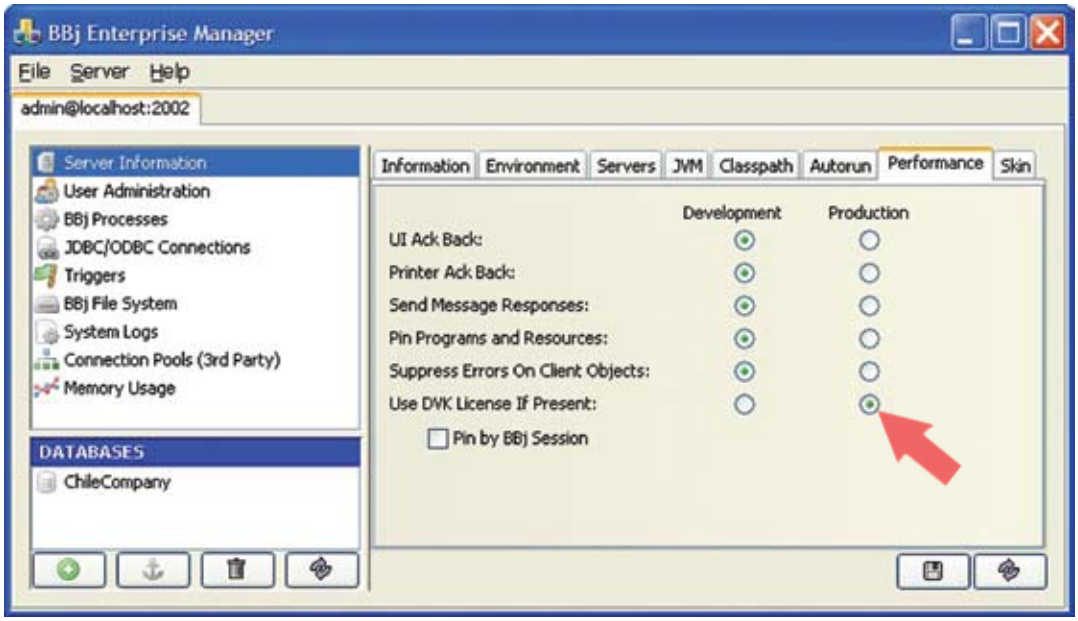

Figure 10. Setting the DVK license for 'Production' mode

Once you are ready to deploy your application, make all your custom jar files available to the server as well as to the client. Add the jar files to the classpath of the server using Enterprise Manager by choosing the Classpath tab under the Server Information section. If you are using Web start to deploy to your clients, then simply add your custom jars in your file.

Summary

ClientObjects allow a BBj program to use arbitrary Java classes on the client. The RMI functionality of ClientObjects provides seamless integration of client-side objects into a BBj program. The BBj program can receive events that the client-side objects have generated and can invoke methods of the client-side objects in the same way that the program invokes methods of server-side objects.

ClientObjects are especially useful for including new GUI classes in a BBj application. Developers can mix-and-match third party GUI components with traditional BBjControls and place these third party controls on a BBjWindow. They can even add BBjControls to third party JComponents.

ClientObjects further reduce the boundary between BBj and Java, allowing developers to access the full power of both languages. They can use BBj for the things that BBj does best and Java for the things that Java does best. In addition, BBj developers are now in a position to take advantage of the large number of third party Java components that are available on the Web. They can integrate anything from a Web browser to a PropertySheet or a DigitalClock into a BBj program with a minimal development effort. ClientObjects, indeed, blow the doors of software development wide open.

Download the code samples from
www.basis.com/advantage/mag-v12n1/clientobjects.zip

Read more about Client Objects at

ClientObject Tutorial
www.basis.com/solutions/ClientObjectTutorial.pdf

BBj Custom Objects Tutorial
www.basis.com/solutions/BBj_CustomObjects.pdf

A Primer for Using BBj Custom Objects
www.basis.com/advantage/mag-v10n1/primer.html

† Download with BBj from www.basis.com/products/bbj/download.html and select "Demos" in the Optional File section. After completing the installation, select **BBj > Demos > LaunchDock** from the BASIS folder to run the demo.