Search

Running BBj Code From Within Java; The Java BBjBridge

By David Wallwork

This article provides an example of the Java BBjBridge and how it can be used to call a BBj® program from within Java code. We will build a BBj program and a Java program that calls it (widget.bbj and WidgetBridge.java). Once this basic structure is in place, we provide two additional programs that demonstrate how a single Java BBjBridge can be reused to provide sub-second response times.

Our BBj program, widget.bbj, accepts an order for widgets and widget batteries. Based on its own somewhat esoteric business rules, it calculates bonus merchandise and dealer discount and returns these values to WidgetBridge.java.

Listing A contains the code for widget.bbj. There are several items in this program that are worth notice.

1. The program has two modes of operation: debug and usingBridge.
2. When running in the usingBridge mode, the program opens a device "J0".
3. When running in the usingBridge mode, the program always terminates by doing a RELEASE.
4. When running in the usingBridge mode, the program tests fid(0) and drops to console if fid(0) is not "IO".

Listing A - widget.bbj (Right click and Save to download code)

```
seterr unknownError
rem ----------------------------------------
rem determine whether we are being called by bridge
rem when not running from bridge we pass the param "debug"
rem ----------------------------------------

fileName$ = "widget.dat"
usingBridge = 1
if argc>1 and argv(1)="debug" then usingBridge = 0

rem ----------------------------------------
rem by default, the terminal for a program called from a
rem bridge is -tIO.  The calling program can set the terminal
rem in BBjBridgeRequest.setCommandLine.  If it has been set
rem to something else then break to console.  This is
rem mechanism to help debug interaction between this program
rem and the calling Java code
rem ----------------------------------------

if usingBridge AND fid(0) <> "IO" then escape
bridgeWithoutConsole:

rem ----------------------------------------
rem open a channel.  if using bridge then config.bbx
rem must contain a line similar to:
rem         "ALIAS J0 com.basis.bbj.bridge.BBjBridgeOpenPlugin"
```

```
rem ----------------------------------------

dataChannel = unt
if usingBridge then
     open(dataChannel)"J0"
else
     open(dataChannel,err=makeTestFile)fileName$
endif

rem ----------------------------------------
rem read order data
rem ----------------------------------------

read record(dataChannel,key="widgetCount",err=readError)widgetCount$
read record(dataChannel,key="batteryCount",err=readError)batteryCount$
read record(dataChannel,key="userID",err=readError)userID$

rem ----------------------------------------
rem use business rules to determine bonusWidgets,
rem bonusMiles and discount
rem ----------------------------------------

rem need to have userID of at least three chars
bonusString$ = userID$
if len(bonusString$) < 3 then bonusString$ = "abc"

bonusWidgets = 1 + mod(asc(bonusString$(1,1)),3)
bonusMiles = 1000 + 1000*mod(asc(bonusString$(2,1)),9)
discount = 5 + mod(asc(bonusString$(3,1)),5)

rem ----------------------------------------
rem write bonusWidgets, bonusMiles and discount to
rem dataChannel
rem ----------------------------------------

bonusWidgets$ = str(bonusWidgets)
bonusMiles$ = str(bonusMiles)
discount$ = str(discount)

write(dataChannel,key="bonusWidgets",err=writeError)bonusWidgets$
write(dataChannel,key="bonusMiles",err=writeError)bonusMiles$
write(dataChannel,key="discount",err=writeError)discount$

rem ----------------------------------------
rem when running in bridge, program may return a result code to
rem the calling Java program.  We return 0 for success.
rem ----------------------------------------

release 0

rem ----------------------------------------
rem error handler if there is a problem reading from dataChannel
rem if usingBridge then do release -1.  By agreement, the
rem Java code will recognize a return code of -1 as a read error
rem ----------------------------------------

readError:
if usingBridge
    release -1
else
    escape
```

```
endif

rem ----------------------------------------
rem error handler if there is a problem writing to dataChannel
rem if usingBridge then do release -2.  By agreement, the
rem Java code will recognize a return code of -2 as a write error
rem ----------------------------------------

writeError:
if usingBridge
    release -2
else
    escape
endif

rem ----------------------------------------
rem error handler for unknown errors
rem if usingBridge then do release -3.  By agreement, the
rem Java code will recognize a return code of -3 as a write error
rem ----------------------------------------

unknownError:
if usingBridge
    release -3
else
    escape
endif

rem ----------------------------------------
rem routine to create file when debugging and no file exists
rem ----------------------------------------

makeTestFile:
mkeyed fileName$, 30, 0, 30
localfile = unt
open (localfile)fileName$
write (localfile,key="widgetCount")"7"
write (localfile,key="batteryCount")"5"
write (localfile,key="userID")"abc"
close (localfile)retry
```

(end of listing)

...Article continued on next page

Running BBj Code From Within Java; The Java BBjBridge (Continued)

By David Wallwork

Two Modes Of Operation

In the first few lines of code, widget.bbj looks for a command line parameter 'debug' and sets a flag usingBridge. The purpose of this is to allow us to test widget.bbj *without* using a bridge. When run using the command "bbj widget.bbj - debug" our program will read input from a local file rather than from the bridge channel. This allows the program to be fully tested before introducing the added complexity of running within a bridge. It is very important that a program that is intended to be called from a bridge should be tested as fully as possible before actually calling it from within Java code since the process of debugging becomes more difficult as we add more 'layers' on top of it.

The "Jxx" Device Type

Just as a device whose name that begins with 'P' always resolves to a printer in BBj, a device whose name begins with 'J' always resolves to a Java plug-in. In our example "J0" is defined in our config file by the line:

```
ALIAS J0 com.basis.bbj.bridge.BBjBridgeOpenPlugin
```

Although a Java plug-in can be provided by the user or by a third party, this particular plug-in is provided by BASIS for the purpose of communicating between Java code and BBj code. Once a channel has been opened with this ALIAS, we can READ data that was placed in the BBjBridgeRequest by the Java code. Any data that we WRITE to that channel will become available to the Java code through the BBjBridgeResponse. For more information see the BASIS Online Documentation.

Calling RELEASE In The BBj Program

The Java program, WidgetBridge.java, can obtain the exit code of widget.bbj in order to determine whether widget.bbj executed successfully. In our example, the Java program and the BBj program have agreed on the meaning of several exit codes. This allows the Java program to provide more meaningful information to the caller in the case that the BBj program fails.

User Interaction When usingBridge Is True

Our goal is to reuse WidgetBridge within a Servlet that will be run on a web server. Since a Servlet has no user interface, we must write widget.bbj in such a way that it will execute without user interaction. So normally, widget.bbj does not drop to console or execute any input verbs. But even if widget.bbj has been well tested, there may be problems in the calling Java code that will cause widget.bbj to fail. If, for example, the calling program has not used the correct key values when writing data to the BBjBridgeRequest, then widget.bbj will fail when it attempts to read the data. Such errors can be difficult to find if we are not able to step through the BBj code while it is being called from the bridge.

We resolve this dilemma by testing fid(0) within widget.bbj. By default, a program that is being executed through a call to Java BBjBridge will have terminal type -tIO. The calling Java program can set a different terminal type by modifying the String that is passed to BBjBridgeRequest.setCommandLine(). In widget.bbj we test to see if the terminal type has been set and if it has then we drop to console. This is a convenient way to allow dot stepping through widget.bbj even while it is being run through Java BBjBridge.

WidgetBridge.java

The code for WidgetBridge.java can be found in Listing B. The main() method in WidgetBridge constructs a WidgetBridge and passes its own command line parameters to widget.bbj. The results of running widget.bbj are printed to System.out and WidgetBridge.main() exits. This main() method may be executed from the O/S command prompt by commands similar to one of the following:

```
java WidgetBridge  123   456  789
```

OR

```
java WidgetBridge  123   456  789  -tT0
```

Although WidgetBridge.java demonstrates how we can use a Java BBjBridge, it is not very efficient. Each time we run WidgetBridge.main() we must:

- start a new Java session
- create a new instance of WidgetBridge
- create a new instance of Java BBjBridge
- establish a new connection to BBjServices

On a reasonably fast machine this will typically take several seconds. In the next section we will examine two programs that reduce this time to the range of 20-40 milliseconds.

Listing B - WidgetBridge.java (Right click and Save to download code)

```
import java.io.*;
import java.text.*;
import java.util.*;
import com.basis.bbj.bridge.*;
public class WidgetBridge{
    // define timeout and expected return codes
    private final static int BRIDGE_TIMEOUT = -1;
    private final static int READ_ERROR = -1;
    private final static int WRITE_ERROR = -2;
    private final static int UNKNOWN_ERROR = -3;

    // define configuration file and program file
    private static String CONFIG_FILE = "c:/basis/work/config.bbx";
    private static String PROGRAM_NAME = "c:/basis/work/widget.bbj";

    // private variables for executing bridge communications
    private JavaBBjBridge m_bridge = null;
    private BBjBridgeRequest m_request = null;
    private BBjBridgeResponse m_response = null;

    // private variables for storing request data and results
    private boolean m_initialized = false;
    private String m_widgetCount;
    private String m_batteryCount;
    private String m_userID;
    private String m_bonusWidgets;
    private String m_bonusMiles;
    private String m_discount;
    private String m_report;
    private String m_consoleString = null;

    /**
     *  It is assumed that the command line parameters to main() will be
     *      widgetCount  --  number of widgets being ordered
     *      batteryCount --  number of batteries being ordered
     *      userID       --  id of user
```

```
 *
 *  It is also assumed that each of these params represents a numeric
 *      value and that userID >= 100
 *
 *  Optionally a fourth parameter of the form -tTxx may be added to allow
 *      dot-stepping through the BBj code during execution
 *
 *  So an expected call line would be
 *
 *        java WidgetBridge 7 3 783
 *  OR
 *        java WidgetBridge 7 3 783 -tT0
 */
public static void
main(String p_argv[])
{
    String resultString = null;

    if(p_argv.length < 3)
    {
        System.out.println("\n\nUSAGE:\n" +
           "       java WidgetBridge widgetCount batteryCount userID\n" +
           "  OR\n" +
           "       java WidgetBridge widgetCount batteryCount userID" +
           " -tTxx\n");             System.exit(-1);
    }
    else
    {
        try
        {
            WidgetBridge widgets;
            if(p_argv.length > 3)
                widgets = new WidgetBridge(p_argv[3]);
            else
                widgets = new WidgetBridge();

            widgets.processOrder(p_argv[0],p_argv[1],p_argv[2]);
            System.out.println(widgets.reportResults());
        }
        catch(BBjBridgeException e)
        {
            e.printStackTrace();
            System.exit(-1);
        }
    }
    System.exit(0);
}

WidgetBridge()
{
    this(null);
}
    WidgetBridge(String p_consoleString)
{
    m_consoleString = p_consoleString;
}

/**
 * processOrder
 */
```

```
public int
processOrder(String p_widgetCount, String p_batteryCount, String p_userID)
    throws BBjBridgeException
{
    m_widgetCount = p_widgetCount;
    m_batteryCount = p_batteryCount;
    m_userID = p_userID;

    int ret = UNKNOWN_ERROR;

    if(checkArgsNumeric())
    {

        // initialize and run the bridge
        initBridge();
        ret = runBridge();
        m_report = createReport(ret);
    }
    else
    {
        m_report = "ERROR: widgetCount, batteryCount and userID " +
                    "must all be numeric";
    }
    return ret;
}

/**
 * reportResults writes the results of running the BBj program
 * to a PrintWriter.
 */

public String
reportResults()
{
    return m_report;
}

/**
 * runBridge() uses current data to run the bridge program and
 * retrieves the results.
 * @ret the release value of the BBj program that was executed
 */

private int
runBridge()

throws BBjBridgeException
{
    // set the program name
    m_request.setProgramName(PROGRAM_NAME);
    // place input data into bridge request
    m_request.put("widgetCount", m_widgetCount);
    m_request.put("batteryCount", m_batteryCount);
    m_request.put("userID", m_userID);
    // run the bridge and obtain it's return value
    int ret = m_bridge.runBBj(m_request, m_response, BRIDGE_TIMEOUT);
    // retrieve the response data from bridge request
    m_bonusWidgets = m_response.get("bonusWidgets");
    m_bonusMiles = m_response.get("bonusMiles");
    m_discount = m_response.get("discount");
```

```
    return ret;
}

/**
 * initBridge() will initialize bridge, request, response if not
 * already initialized.  initBridge() is no-op if already initialized
 * @param p_consoleString can be be a string of the form "-tT0" to
 * specify a terminal to be used for debugging the BBj code while
 * it is running in this WidgetBridge.  If this param is null or if it
 * an empty string then the string should represent a valid Terminal ALIAS
 * present in the config file.
 */

private void
initBridge() throws BBjBridgeException
{
    if( ! m_initialized)
    {
        if(m_consoleString == null)
            m_consoleString = "";
        String cmdLine = " -c" + CONFIG_FILE + " " + m_consoleString;
        m_bridge = JavaBBjBridgeFactory.createBridge(
                                  cmdLine,true,true,false,true);
        m_request = JavaBBjBridgeFactory.createBridgeRequest();
        m_response = JavaBBjBridgeFactory.createBridgeResponse();
        m_initialized = true;
    }
}

/**
 * checkArgsNumeric checks that input params are all numeric
 */

private boolean
checkArgsNumeric()
{
    boolean ret;
    try
    {
        // call parseInt to ascertain that all params are numeric
        Integer.parseInt(m_widgetCount);
        Integer.parseInt(m_batteryCount);
        Integer.parseInt(m_userID);
        ret = true;
    }
    catch(NumberFormatException e)
    {
        ret = false;
    }
    return ret;
}
private String
createReport(int p_returnValue)
{
    String ret;
    // write results to p_out
    switch(p_returnValue)
    {
        case 0:
           // print results to stdout
           ret =  "\n\nTest Results from running Widget test: " +
```

```
                    "\n       bonus Widgets:  " + m_bonusWidgets +
                    "\n       bonus miles:    " + m_bonusMiles +
                    "\n       discount:       " + m_discount;
                break;
            case READ_ERROR:
                ret = "Bridge program experience error reading";
                break;

            case WRITE_ERROR:
                ret = "Bridge program experience error reading";
                break;

            case UNKNOWN_ERROR:
            default:
                ret = "unknown error executing bridge program.";
                break;
        }
        return ret;
    }

    public void
    finalize()
    {
        m_bridge.close();
    }
}
```

(end of listing)

...Article continued on next page

Running BBj Code From Within Java; The Java BBjBridge (Continued)

By David Wallwork

WidgetTimer.java And WidgetServlet.java

WidgetTimer.main() (see Listing C) creates a WidgetBridge and makes multiple calls to processOrder(). It prints the elapsed time for each execution of processOrder(). Since we use the same Java BBjBridge() repeatedly, this execution time becomes quite small (20-40 milliseconds). WidgetTimer can be run with a command line similar to:

```
java WidgetTimer   222   333    444
```

Listing C - WidgetTimer.java (Right click and Save to download code)

```
import java.io.*;
import java.text.*;
import java.util.*;
import com.basis.bbj.bridge.*;

public class WidgetTimer{
    public static void main(String p_argv[])
    {
        if(p_argv.length < 3)
        {
            System.out.println("\n\nUSAGE:\n" +
               "       java WidgetTimer widgetCount batteryCount userID\n");
            System.exit(-1);
        }
        else
        {
            try
            {
                WidgetBridge widgets = new WidgetBridge();
                for(int q=0; q<10; ++q)
                {
                    long start = System.currentTimeMillis();
                    widgets.processOrder(p_argv[0],p_argv[1],p_argv[2]);
                    long end = System.currentTimeMillis();
                    System.out.println("WidgetBridge.processOrders() " +
                            " executed in " + (end - start) + " mSec");
                }
            }
            catch(BBjBridgeException e)
            {
                e.printStackTrace();
                System.exit(-1);
            }
        }
```

```
        System.exit(0);
    }

}
```

Our final code sample is WidgetServlet.java (see Listing D), an HTTPServlet that can be deployed in a Servlet-aware web server. Again, it re-uses the same Java BBjBridge and so provides fast execution times. WidgetServlet.doGet() uses parameters from the HTTPServletRequest to call widget.bbj and prints the results to the OutputWriter of the HTTPServletResponse.

Listing D - WidgetServlet.java (Right click and Save to download code)

```
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.basis.bbj.bridge.*;
public class WidgetServlet extends HttpServlet
{
    WidgetBridge m_widgetBridge;

    /**
     * doGet() will be called by the Servlet-aware web server in response
     * to a GET request.  It will generate HTML containing the results of
     * calling the program widget.bbj with the parameters passed into the
     * GET request.
     */

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        String title = "demo of JavaBBjBridge within Servlet";
        out.println("<title>" + title + "</title>");
        out.println("</head>");
        out.println("<body bgcolor=\"white\">");
        out.println("<body>");
        out.println("<h1>" + title + "</h1>");

        // retrieve the input params from request
        String widgetCount = request.getParameter("widgetCount");
        String batteryCount = request.getParameter("batteryCount");
        String userID = request.getParameter("userID");
        if( (widgetCount == null) || (batteryCount == null) ||
                (userID == null) )
        {
            out.println("\n\n ERROR:  Missing argument.  Request must contain " +
                " widgetCount, batteryCount and userID");
        }
        else
        {


           // initialize and run the bridge
           try
           {
```

```
                synchronized(m_widgetBridge)
                {
                    m_widgetBridge.processOrder(
                                    widgetCount, batteryCount, userID);
                    out.println(m_widgetBridge.reportResults());
                }
            }
            catch(BBjBridgeException e)
            {
                out.println("ERROR: unknown exception processing order:\n\t" +
                            e.getMessage());
            }
        }
        out.println("</body>");
        out.println("</html>");
    }

    /**
     * init() is called when this Servlet is first instantiated.  It
     * creates a new m_widgetBridge that can be used thoughout the
     * lifetime of this Servlet
     */

    public void init()
    {
        m_widgetBridge = new WidgetBridge();
    }

    /**
     * destroy is called when this Servlet is terminated.
     * It closes the m_widgetBridge
     */

    public void destroy()
    {
        m_widgetBridge.finalize();
        m_widgetBridge = null;
    }
}
```

Try It!

These samples should provide a starting point for the reader who wants to call BBj code from within Java. It is especially important when using the bridge that developers first debug their BBj code before attempting to call it from a Java BBjBridge. But if your BBj code is well tested then implementing the Java code that can be used to call your BBj program is rather simple.

Although the initial cost of creating a Java BBjBridge is significant, it can be reused to access your BBj code with minimal overhead.

Calling BBj from within Java is really very easy. Try it!