

Confessions of a Language Polygamist

By Jon Bradley

Some programmers continually experiment with new languages and feel comfortable using multiple languages, even on the same day. I am not one of those people. I have always regarded myself as a *language monogamist*. In the same way that some people can chart their lives by the music albums they listened to in different periods of their lives, I can divide my life into stages based on the language in which I programmed. As a young rookie, I programmed in QBasic, then C. Somewhere in college I began to experiment, going from C to PERL. After a summer job tricked me into switching from PERL to Java, I thought Java was my “one and only - ‘til death us do part.”

My Confession

Well I was wrong. Over the course of several years as a BASIS Java engineer, BBJ® began to move ahead and take its place as my second workhorse language. It all came as a bit of a surprise, but often BBJ was the fastest way I could think of to implement some random application, be it a test harness, or a mortgage calculator. In many ways, BBJ meets or beats Java by providing a sophisticated environment, an easy to use GUI, and some great development tools. This makes sense when you stop to think about it. BASIS wrote BBJ in Java. The BBJ project has many hundreds of thousands of lines of code; many of which specify improvements in Java functionality, or simplify tasks that are more difficult in Java.

At BASIS’ TechCon, an attendee asked me “If I don’t have any old PRO/5® code or data and if I’m starting from scratch, why would I use BBJ instead of just using Java?” It then occurred to me that we do not highlight the improvements BBJ makes upon Java. So, I took the mission upon myself to do just that – expose, in this article, the many ways that developing in BBJ is BETTER than or AS good as developing in Java.

Simpler Code

In many cases, BBJ code is simpler and shorter than Java. For instance, compare the standard **Hello <name>** program in BBJ (Figure 1) with Java (Figure 2):

continued...

```
input "Please type your name: ",name$
print "Hello ",name$
```

Figure 1. **HelloName.src** code sample in BBJ

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class HelloName
{
    public static void main(String args[]) throws IOException
    {
        String name;
        BufferedReader br = new BufferedReader(
            new InputStreamReader(System.in));
        System.out.print("Please type your name: ");
        name = br.readLine();
        System.out.println("Hello "+name);
    }
}
```

Figure 2. **HelloName.java** code sample in Java



Partnership

Language/Interpreter

Database Management

Development Tools

System Administration



Jon Bradley
Software
Engineer

An even more revealing example is in a small GUI application illustrated in the next two figures. Compare the incredibly large amount of complex Java code in **Figure 3** with the smaller amount and far simpler BBj code in **Figure 4**. Two very different programs, one almost triple the size of the other, result in identical output.

continued...

```
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.UIManager;

public class PushMe extends JFrame
{
    public static void main(String args[])
    {
        //kind of optional, but i wanted the two apps to look the same
        setupLookAndFeel();

        JFrame win = new JFrame();
        win.setTitle("Simple GUI App");
        win.setPreferredSize(new Dimension(200,200));
        win.setDefaultCloseOperation(win.EXIT_ON_CLOSE);
        win.getContentPane().setLayout(null);
        JButton button = new JButton("Push Me");
        button.addActionListener(new DialogPopper(button,win));
        button.setSize(new Dimension(120,30));
        button.setLocation(30,50);
        win.getContentPane().add(button);
        win.pack();
        win.setVisible(true);
    }

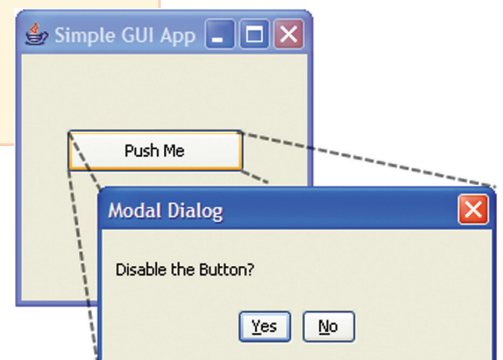
    /** Not necessary, but i wanted the programs to look the same,
     * and Java by default doesn't do the XP look and feel
     */
    private static void setupLookAndFeel()
    {
        try {
            UIManager.setLookAndFeel(
                UIManager.getSystemLookAndFeelClassName() );
        } catch (Exception e) { }
    }
}

class DialogPopper implements ActionListener
{
    private final JButton m_button;
    private final JFrame m_frame;

    DialogPopper(JButton p_button, JFrame p_frame)
    {
        m_button = p_button;
        m_frame = p_frame;
    }

    public void actionPerformed(ActionEvent p_e)
    {
        int result = JOptionPane.showConfirmDialog(m_frame,
            "Disable the Button?", "Modal Dialog",
            JOptionPane.YES_NO_OPTION, JOptionPane.PLAIN_MESSAGE);
        if(result == JOptionPane.YES_OPTION)
        {
            m_button.setEnabled(false);
        }
    }
}
```

Figure 3. PushMe code written in Java to create the sample GUI output



```

SYSGUI=UNT
OPEN(SYSGUI)"X0"

declare BBjWindow win!
rem height is different because our height specifies client height, not frame height
win! =BBjAPI().getSysGui().addWindow(50,50,200,170,"Simple GUI App")
win!.setCallback(win!.ON_CLOSE,"APP_CLOSE")

declare BBjButton button!
button! = win!.addButton(101,30,50,120,30,"Push Me")
button!.setCallback(button!.ON_BUTTON_PUSH,"PUSHED")

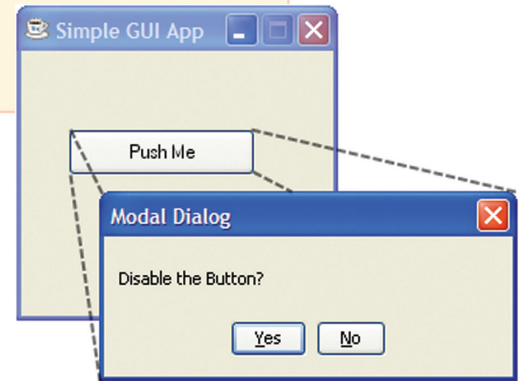
PROCESS_EVENTS

PUSHED:
if(MSGBOX("Disable the Button?",4,"Modal Dialog") = 6) button!.setEnabled(0)
return

APP_CLOSE:
release
return

```

Figure 4. PushMe code written in BBj to create the same sample GUI output



As you can see from these small examples, the BBj code in **Figure 4** is more succinct. BBj provides a host of language functionalities that other languages, such as Java, typically leave to the developer. For instance, Databound GUI controls make the process of displaying and updating table/file information significantly less complicated in BBj.

Objects

An area in which BBj used to lag was its object oriented capabilities. Since 2006, BASIS has offered [custom objects](#) that provide the developer with a robust object system. In addition to custom objects, BBj has always been able to embed Java objects including the extensive core libraries, or custom Java classes.

Database Management System

Out of the box, Java does not provide any keyed table or file types or database capabilities. BBj, however, provides transaction tracking, extremely fast keyed look-ups, table/file locking, and full DBMS functionality. BBj provides SQL access via both the SELECT verb, and JDBC/ODBC access. BBj provides [triggers](#) and [stored procedures](#) to ease implementation of a host of data-centric issues. In addition, both trigger and stored procedure logic is coded with the BBj language.

Deployment Choices

Using 3-tier [BASIS architecture](#), developers can deploy BBj in many different ways. Thin Client allows the data and processing to occur on a remote server, while the GUI or CUI presentation occurs on the client machine. Deploying through Web Start can update client machines automatically with the latest BBj.

Interactive Interpreter

In BBj, the developer can debug/edit a running application within our fully interactive interpreter. This greatly speeds development and maintenance of applications due to the ability to load or run arbitrary code at the command prompt. There simply is no **>READY** prompt in Java. For many of our developers this feature has been the essential enabler for delivering exceptional end-user support and troubleshooting capability.

Program Format

BBj can load or save both ASCII and tokenized programs. Furthermore, a developer can load a program as ASCII and save it as tokenized, or vice versa. In Java, not only is the source program always flat text, (with a required **.java** extension), but Java can only run a compiled **.class** file. BBj also provides SAVEP protection to protect your source code from prying eyes.

continued...

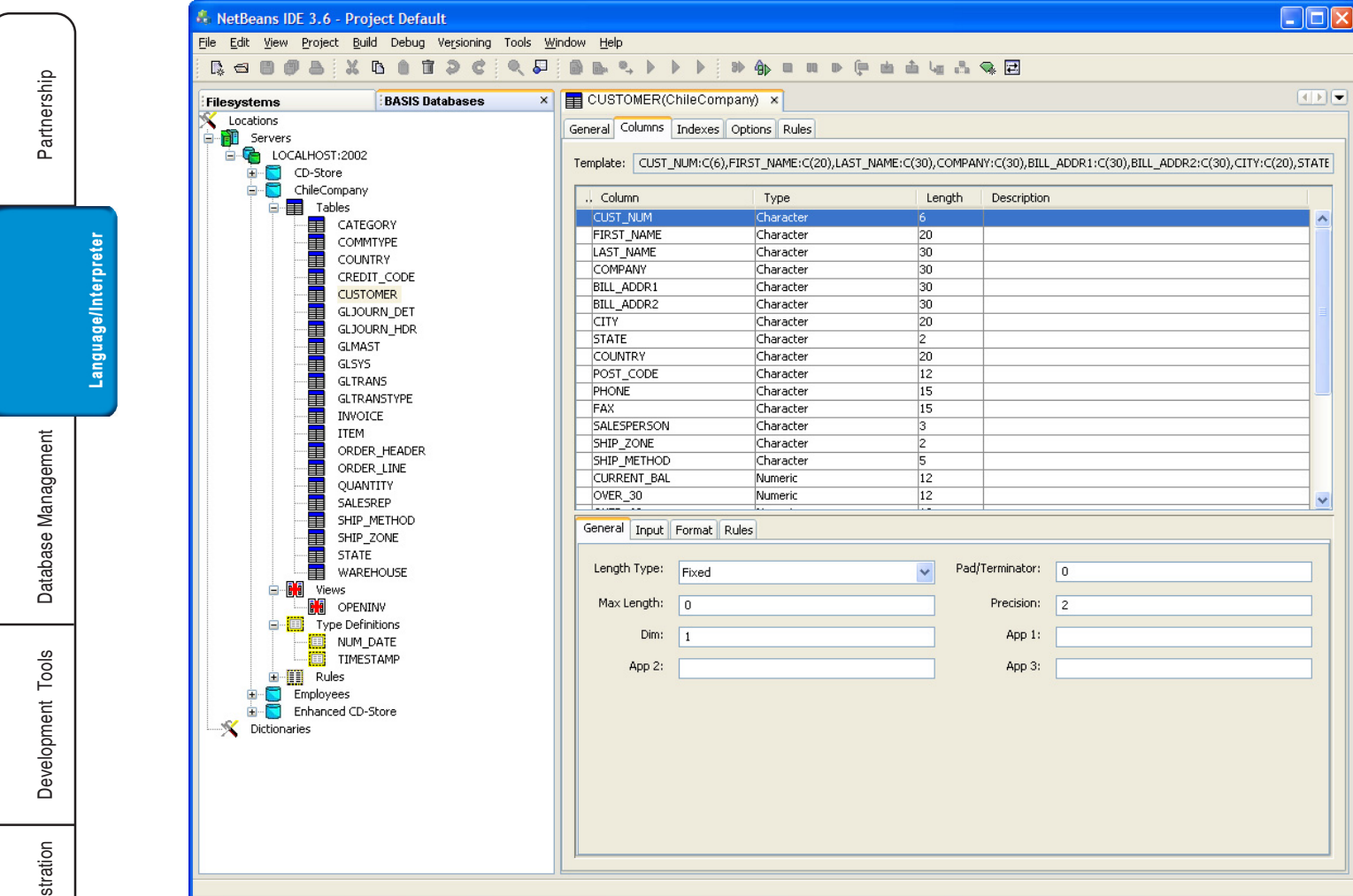


Figure 5. Configuring a database using the BASIS IDE


Developer Tools

BASIS facilitates development of applications with the [BASIS IDE](#) as shown in **Figure 5**. From within the IDE, you can [edit and debug programs](#), configure [Data Dictionary](#) tables/files, [create resource files](#) to specify GUI forms, [view data](#) within BASIS-keyed tables/files, or quickly create full applications using [AppBuilder](#).

Developer Support

Anyone active on the BBJ developer list knows that BASIS provides very responsive support to developers. While Java does have a developers' forum, you can measure responsiveness to a bug report in 'months.' BASIS often provides turnaround from a support and sales standpoint the same day. Our engineering department frequently incorporates suggestions from the development community in upcoming releases.

Summary

So, it is no secret that I still program in Java, but there are so many situations where BBJ's interactive interpreter, easy table/file access, and GUI system make it the easiest language to use. The BASIS IDE allows me to create GUI screens and full applications faster than I could in Java. With backward compatibility to legacy Visual PRO/5[®] code and data, BBJ certainly has its own share of strengths when standing up against Java. I guess this does make me a language polygamist...enjoying the best of both worlds. 



For additional information, read *Using Triggers to Maintain Database Integrity* on page 6 and *Using Stored Procedures to Add Business Logic to the Database* on page 8 in this issue.

For more information about deployment choices, read *Choices, Choices, Choices* at www.basis.com/advantage/mag-v9n2/choices.html