From BBx to BBj
Techniques for Migrating And Optimizing Your Application

By Jim Douglas

In the Q3/Q4 2000 issue of the BASIS International Advantage, we talked about the lessons we learned in developing our b-CommerceT application in GUIBuilder® and deploying it over the Web using a prerelease version of BBj®. Now that BBj 1.0 is a released product, we can describe the process of taking an existing PRO/5® or Visual PRO/5® application and moving it to BBj.

What is BBj?

Our primary goal was to make BBj as compatible as possible with BBx®. We've succeeded in this goal. The vast majority of existing BBx programs run unchanged, or with very minor changes, in BBj. But if BBj were just compatible with BBx without offering compelling new benefits, you would have no reason to move your applications from BBx to BBj. As a true three-tier product, BBj enables to you to distribute the business rules, database and user interface in whatever way makes sense for your application. In particular, you can deploy existing character or GUI applications over the Web. BBj unifies the platform-specific features of PRO/5 (pipes and the BACKGROUND verb) and Visual PRO/5 (GUI and SQL support) in a single cross-platform environment. And many of the inherent limits of BBx have been removed: BBj supports arbitrary-precision numbers; variable names and template field names are no longer limited to 32 characters; mnemonic string parameters are no longer limited to 250 bytes; and the new XKEYED file type eliminates all of the MKEYED file limits.

What's Different?

While BBj is a Java-based product, BBx is a C++-based product. This change in platforms forced a few language changes. For example, BBj doesn't support C-ISAM. The C-ISAM file type is implemented in BBx by linking "C" libraries from Informix, and these libraries aren't available in Java. Also, Windows-specific features (in particular, DDE) aren't available in BBj.

We were able to make many improvements in the BBj language, with the most important being the introduction of reserved keywords. The BBx language specification has several ambiguities. For example, all of the following lines of code are accepted in BBx:

```
1010 LET TEMP$=DATE(JUL(Y,M,D):FMT$),ERR=1090
1020 REM: LET X=X+1
1030 IF THEN THEN GOTO 1040
1040 LET CTL=0
```

BBx interprets these lines differently from what the programmer intended, creating subtle and hard-to-spot errors. BBj flags all of them as syntax errors. BBj comes with a new utility, _keyword, which scans BBx programs for reserved keywords being used as numeric variables or labels. This scan will highlight the errors in lines 1010 (ERR=1090), 1020 (REM), 1030 (THEN) and 1040 (CTL).

We've implemented the BBj parser using a formal language grammar. This grammar-based approach catches potential errors that slip through the BBx parser. For example, BBx accepts the following two lines of code, generating run-time errors when they are executed:

```
1050 LET X$=X$(1,2,ERR=3)
1060 LET X$=X$(X.2)
```

Because BBj flags these lines as syntax errors as soon as they are entered, your code is more likely to be error-free from the start. You should recompile your code up front using the BBj compiler (bbjcpl). Recompiling in BBj will make your programs faster (because BBj loads BBj-tokenized programs faster than BBx-tokenized programs), and it will locate syntax errors like the ones shown above.

How to Get There

Here are the steps that you should follow to convert your application from BBx to BBj.

1. Run the _keyword utility. Whenever it finds a reserved keyword being used as a numeric variable or label, it will append an underscore to the name. For example, the lines from above would become:

   ```
   1010 LET TEMP$=DATE(JUL(Y,M,D):FMT$),ERR_=1090
   1020 REM_: LET X=X+1
   1030 IF THEN_ THEN GOTO 1040
   1040 LET CTL_=0
   ```

   The _keyword utility also has the option to just produce a report without making any changes to your programs.

2. Recompile your programs using bbjcpl.
3. If this is a Visual PRO/5 application that uses resource files, convert the resource files from binary (*.brc) to ASCII (*.arc) using either ResConverter or ResBuilder®.
4. Review and test the code.

Some applications will be affected by the following language changes:

1. BBj can load and run programs in BBx PROGRESSION/4® or PRO/5 token format, but BBj uses its own token format that isn't compatible with BBx. Any code that depends on BBx tokens, especially the CPL() and LST() functions, will need to be modified for BBj.

2. The INFO(0,0) function returns the name of the operating system as reported by the Java Runtime Environment. For example, typical return values in BBj for the various versions of Microsoft Windows are "Windows 95," "Windows 98," "Windows NT," and "Windows 2000," as opposed to "WIN95," "WIN98," and "WIN/NT," which BBx returns.

3. The INFO(6,0) function returns the name of the underlying GUI. BBx returns "WINDOWS" while BBj returns "SWING." Both BBx and BBj return INFO(6,0)="" if no GUI is available.

4. The FID() function reports a file type of 84 for BBj-tokenized program files.

Modify any code that tests for file type 04 to identify a program to test for either 04 or 84.

5. Because the BBj workspace is dynamic, code that tests DSZ for a particular range and then does a START of the current program should be disabled. In many cases, this sort of code can generate an infinite loop in BBj.

For a complete list of language changes, see the ***Converting to BBj and Language Changes*** sections of the BBj documentation, especially ***Converting to BBj from Earlier Versions of BASIS Products and Miscellaneous Language Changes.***