Speeding Up The Development Process - The BASIS IDE

By Michael Phelps

Software development has come a long way since the days of the **READY>** prompt. The dawn of the Graphical User Interface (GUI) spawned sweeping changes. Greater sophistication and convenience for the user means tremendously increased complexity for the developer, whose old methods and character-based tools are quickly becoming inadequate. Years ago, BASIS offered a full screen editor utility, _edit, with features like built-in syntax checking, search and replace, and hotkeys for merging subroutines. Command line-driven lister and compiler utilities followed, which allowed business BASIC developers to choose their own third-party text editors. BASIS's first truly graphical product, Visual PRO/5®, offered ResBuilder®, GUIBuilder® and DDBuilder® to help develop applications for Microsoft Windows®.

With the advent of BBj®, BASIS faced a crossroads with the realization that stable, separate, and standalone utilities were inadequate. Accessing the power of BBj and using it to write graphical, multi-platform, Web-enabled applications demanded something much better. BASIS needed a new tool, designed from the ground up, to work with GUIs and still handle the character-based code of the past. Furthermore, BASIS needed a development system to bring together the many processes required to build a modern software application. BASIS needed a complete Integrated Development Environment (IDE).

BASIS decided that the fastest and best way to create such a development environment was to take advantage of the work of those who already solved these problems. The answer was NetBeans®, an open-source Java-based IDE project sponsored by Sun Microsystems, the developers of Java itself. BBj 2.0 introduced BASIS's first NetBeans-based IDE for writing Business BASIC software using BBj. Although this first IDE had limitations, it was a quantum leap beyond anything previously offered. For the first time, a dedicated source code editor contained features such as syntax highlighting, abbreviation expansion, and keyboard macro recording.

BASIS also integrated the WinConsole debugger into the IDE, making it possible to write, debug, and execute BBj code within a single tool. ResBuilder, GUIBuilder, and DDBuilder are launched from within the IDE. Since that first release, both NetBeans and Java have moved forward. We at BASIS have also been busy, and we know that you will enjoy the new capabilities of the BBj 3.01 IDE.

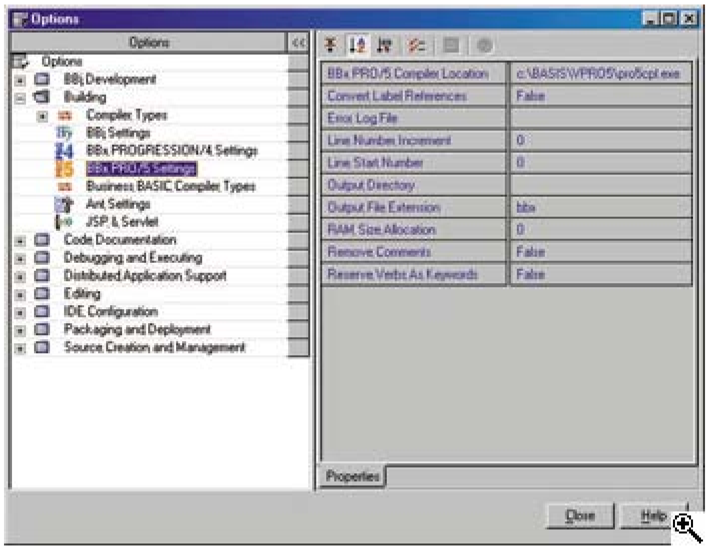
What follows is a brief look at some of the very exciting features first introduced at BASIS TechCon2003.

"Pluggable" Business BASIC Compilers

Perhaps the greatest enhancement to the IDE in BBj version 3.01 is its ability to compile Business BASIC source text files into tokenized files from within the IDE. If developers have BBx® PROGRESSION/4 or BBx PRO/5® installed, they can now compile tokens for those versions, as well as for BBj. The IDE is no longer a "BBj only" development environment!

To display the list of compiler options, open the Options window and expand **`Options -> Building`**. Then, select which type of Business BASIC to work with. The three choices available from the Business BASIC Compiler Types option are BBj, BBx PRO/5, or BBx PROGRESSION/4. **Figure 1** shows the BBx PRO/5 settings, each of which is delivered as a parameter to **pro5cpl** at the start of a compile. Simply enter the path to the **pro5cpl**, select any combination of files or directories in the Explorer, and select Compile or Build from the Build menu to start compiling.

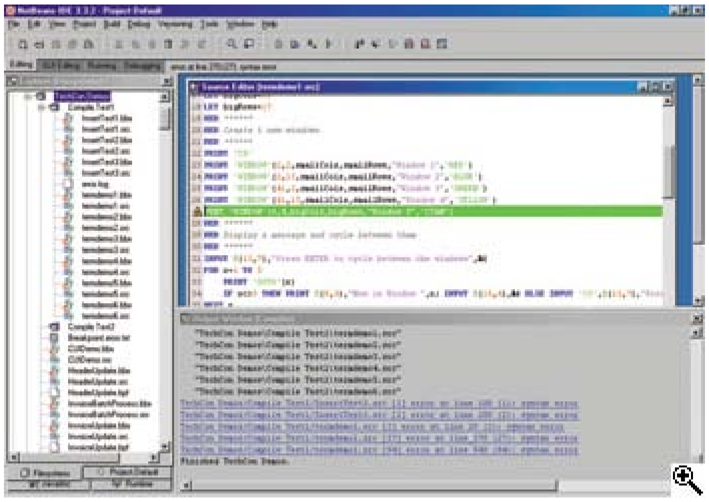
Figure 1. BBx PRO/5 compiler settings.



BBx PRO/5 Compiler Location	c:\BASIS\PRO5\pro5cpl.exe
Convert Label References	False
Error Log File	
Line Number Increment	0
Line Start Number	0
Output Directory	
Output File Extension	bbx
RAM Size Allocation	0
Remove Comments	False
Reserve Verbs As Keywords	False

Figure 2 displays the results of turning the TechCon Demos directory into PRO/5 tokens, using Compile All (or Build All) rather than Compile or Build. "All" means compile or build recursively. It compiles all files in all the subdirectories of the directory already selected in the Explorer. Specifying an output directory in the options places the tokenized files in a directory structure that corresponds to the structure of the source directory tree.

Figure 2. Compiler results for the TechCon demo programs.



Because the **.bbx** extension was associated with PRO/5 tokenized files in the compiler options, the Explorer shows a new icon in front of the compiled program names. This clearly distinguishes them from the original text files.

Notice the Output Window in **Figure 2.** This window shows a compiler-generated list of the files it is compiling. If the developer did not specify the sending of errors to an error log file, the compiler displays errors as hyperlinks. These hyperlinks take the developer directly to the line containing the error in the Source Editor. In this particular case, the file **TechCon Demos/Compile Test1/termdemo1.src** contained a syntax error on line 27. Double-clicking on the hyperlinked error message opened **termdemo1.src** in the Source Editor and highlighted line 27, where an obvious typo in the PRINT verb exists.

Compiling in BBx PROGRESSION/4 and BBj works the same way. In fact, using the IDE to develop in BBj allows integrated compiling. This means the developer can now see syntax errors in the Source Editor without ever having to go to the Debugger or start a BBj interpreter. It is no longer necessary to load the files into the IDE Source Editor or Debugger, one at a time, and individually save them as tokens. Additionally, it is no longer necessary to produce and maintain complex scripts to perform recursive batch compiling. To develop in another version of the Business BASIC language, simply select the desired compiler type. To compile, select everything required in the Explorer and then select Build All. If there are compiler errors, click on the links to quickly see the errors.

Persistent Breakpoints

BBj 3.01 provides an important new debugging capability. When trying to hunt down an obscure problem with the IDE Debugger, it is useful to plant breakpoints in the code so that execution will pause at specific places, where the state of the program can be checked. Until now, adding breakpoints to code was only possible with code already visible in the Debugger. Furthermore, the breakpoints disappeared when the developer closed or exited the file. This made it difficult or impossible to use breakpoints for troubleshooting problems in applications with many CALLed levels.

The BASIS IDE now allows the insertion of breakpoints in the Source Editor and Debugger. The IDE automatically saves the breakpoints to disk when saving the source code.

Figure 3 shows **calltest.bbj** and four other programs with nested CALL statements loaded into the Source Editor. The **calltest2.bbj** and **calltest4.bbj** programs each contain a breakpoint, set in the same way that the Debugger creates breakpoints. Saving these two files also automatically saved the breakpoints to disk files. In the Explorer to the left of **Figure 3**, notice that **calltest2.bbj** has a matching breakpoint file with the name **calltest2.bpf**, and **calltest4.bbj** has **calltest4.bpf**. Whenever a file is loaded into the Source Editor or Debugger, the IDE checks to see whether a **.bpf** file exists in that same directory. If the file exists, the breakpoints are then loaded and displayed along with the text of the program.

Figure 3. Breakpoints in the Source Editor.

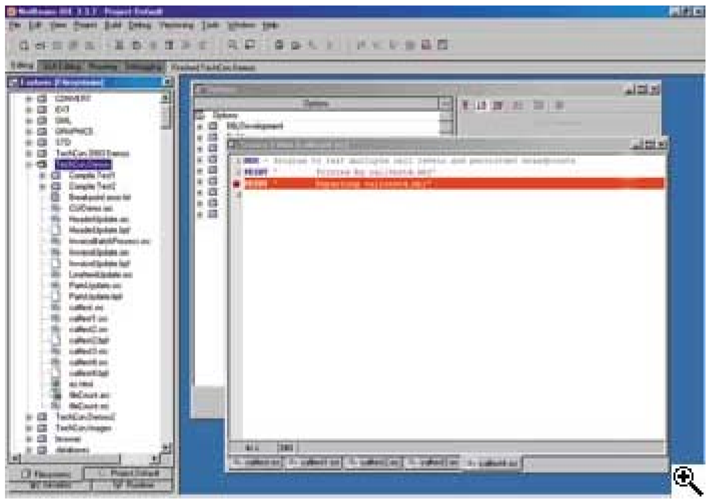
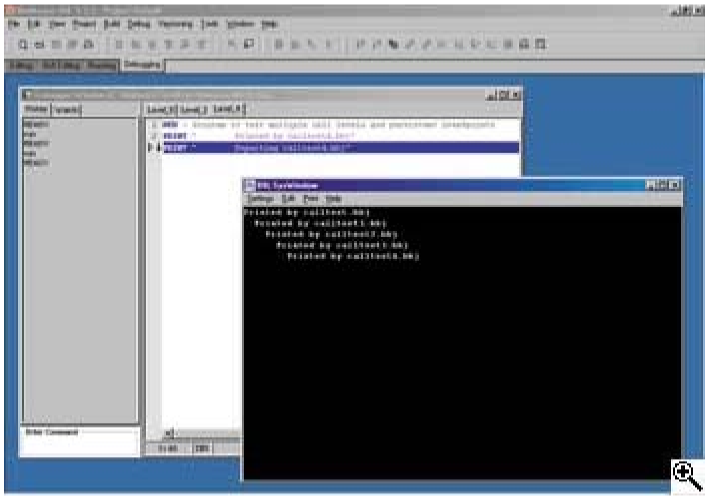


Figure 4 shows the result of executing **calltest.bbj** in the Debugger and then

entering RUN more than once at the command line. The first RUN jumped out to the breakpoint in **calltest2.bbj**, causing the Level_2 tab to appear, and the second RUN jumped to the breakpoint in **calltest4.bbj**, which the Level_4 tab displays. Setting breakpoints in the Source Editor automatically makes them available to the Debugger. In this instance, the breakpoints allowed the Debugger to skip over all the code in **calltest1.bbj** and **calltest3.bbj**. This ability to save breakpoints allows developers to devise more complex debugging schemes, which are preserved between debugging sessions.

Figure 4. Result of executing **calltest.bbj** in the Debugger.



```
Printed by calltest.bbj
  Printed by calltest1.bbj
    Printed by calltest2.bbj
      Printed by calltest3.bbj
        Printed by calltest4.bbj
```

In addition, BBj 3.01 features an enhanced command line in the IDE Debugger and standalone WinConsole. The command history now saves all commands entered on the command line with these tools. Pressing the up and down arrow keys recalls and re-executes previously entered commands. The CTRL ALT up arrow or CTRL ALT down arrow key combinations now cause a single "dot step" command that is sent to the BBj interpreter. This is more convenient than typing a decimal character and then pressing ENTER.

Source Code Version Control

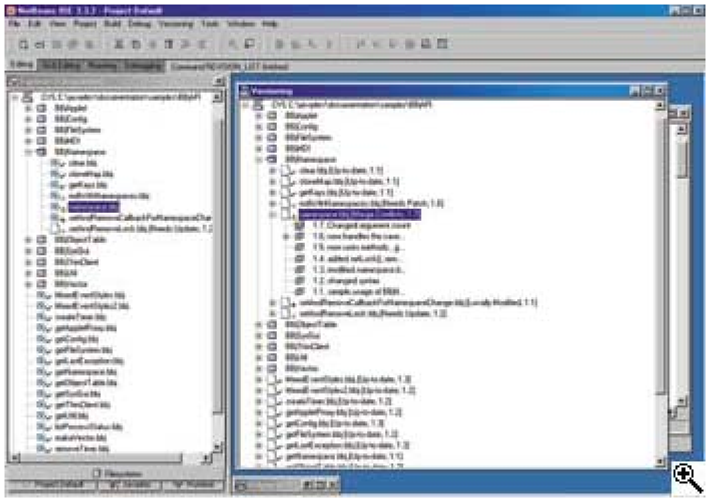
A source code Version Control System (VCS) is an essential component of modern software development, and no Business BASIC developer should be without one. Simply stated, a VCS is a central repository of files containing the source code and resources belonging to a software project, organized in whatever directory structure a development team desires. Individual developers check out files from the central archive, make and test their changes, and then return the altered files to the repository. The VCS manages the check-out/check-in process and keeps track of which versions of files belong to which versions of the final product. Such a system imposes order on the software development process by allowing developers to:

- Effortlessly share the same code base, quickly communicating changes to each other
- Automatically merge several developers' changes
- Instantly restore accidentally deleted files or directories
- Trace changes made to a file in the repository back to the exact location, to the developer who made the change, purpose of the change, and the date of its check-in to the repository
- Maintain older and re-released versions of the software project without affecting work on the most current version
- Quickly "roll back" the project to a previous condition, if necessary, due to a fatal software bug or other problem

The BASIS IDE makes it easy to work with version control systems that have a command-line interface. A built-in graphical client provides a GUI interface for operating the system inside the IDE without the need for developers to use a separate tool or master a sophisticated command line syntax.

Figure 5 shows directories and files belonging to a repository created with the Concurrent Versioning System (CVS), which is available at no cost on the Internet at ***www.cvsnt.org/wiki*** or ***www.cvshome.org/docs/manual/***. A network administrator created the repository on a networked server after installing and configuring CVS. On the left side of this figure, notice the mounting of the repository directories in the Explorer and the expansion of the directory called BBjNamespace, showing what is happening to its files.

Figure 5. CVS repository of source files.



A check mark to the left of the file name indicates that **clear.bbj**, **cloneMap.bbj** and **getKeys.bbj** are up-to-date. In other words, the checked out copies of these files on the local machine are identical to the copies stored in the repository. However, one can see that there are differences in the other files. The diamond symbol beside the **MDIWithNamespaces.bbj** icon means another developer modified this file and then checked a new version into the repository. The developer must now check out a new version of this file to stay in sync with the rest of the development team.

The plus symbol next to the icon for **setAndRemoveCallbackForNamespaceChange.bbj** indicates that the version of this file on our local machine is different from the version in the repository. If these differences are worth keeping, commit the file to the repository so that the updates are available to everyone else on the development team.

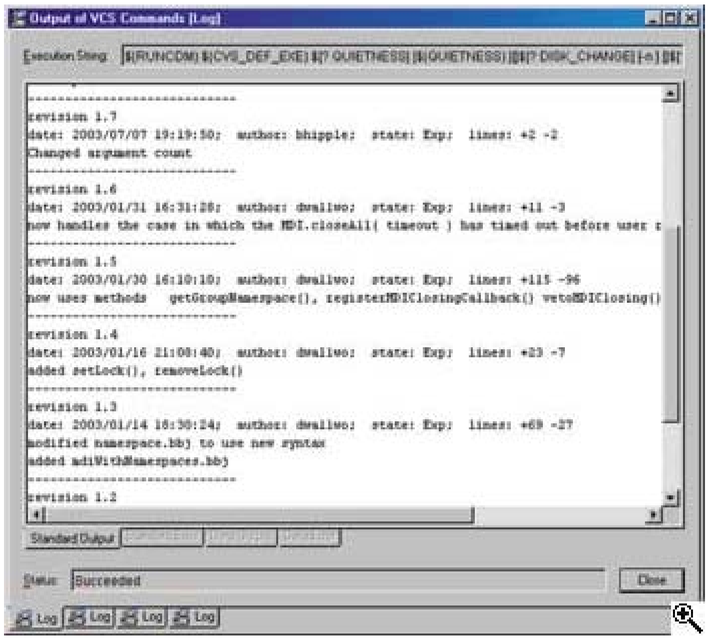
The file named **setAndRemoveLock.bbj** also has a diamond, but the normal BBj file type icon is missing. This means that the repository contains this file, but our local machine does not have a working copy checked out.

A more serious situation exists with the file called **namespace.bbj**. The triangle symbol with the exclamation mark indicates that another developer tried to alter the same lines in the file. Before checking this file back into the repository, the developer must resolve this conflict.

The Versioning Explorer gives specific information about the **namespace.bbj** file. There are seven different versions of the file in the archive, numbered 1.1 to 1.7. This means that a developer made changes six times to this file since its creation. The conflict appeared when someone attempted to create an eighth version of the file.

Asking to see the log for the **`namespace.bbj`** file produces the window shown in **Figure 6**. A software developer by the name of dwallwo wrote the first six versions of this file. His comments about the file's changes appear at the bottom of each separate version. Because bhipple created the latest version of **`namespace.bbj`**, the conflict concerns bhipple's change.

Figure 6. CVS Log.



Output of VCS Commands [Log]

Execution String: $(RUNCDM) $(CVS_DEF_EXE) $[? QUIETNESS] [$(QUIETNESS)] [$[? DISK_CHANGE] [-n] $[

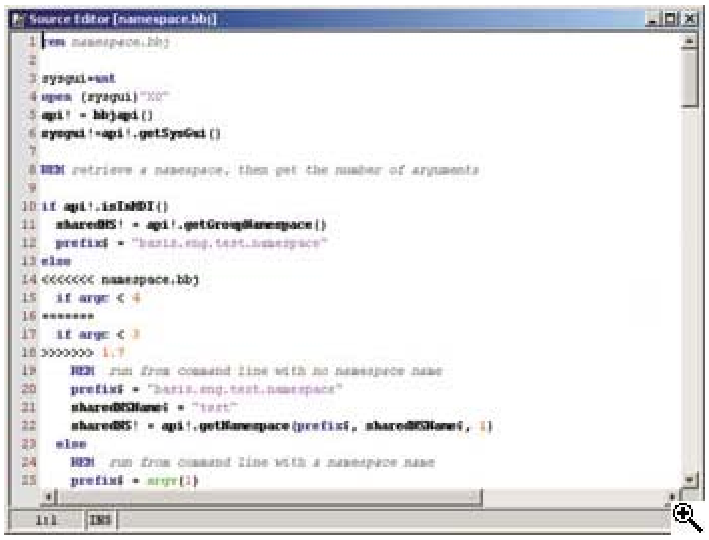
```
----------------------------
revision 1.7
date: 2003/07/07 19:19:50;  author: bhipple;  state: Exp;  lines: +2 -2
Changed argument count
----------------------------
revision 1.6
date: 2003/01/31 16:31:28;  author: dwallwo;  state: Exp;  lines: +11 -3
now handles the case in which the MDI.closeAll( timeout ) has timed out before user r
----------------------------
revision 1.5
date: 2003/01/30 16:10:10;  author: dwallwo;  state: Exp;  lines: +115 -96
now uses methods   getGroupNamespace(), registerMDIClosingCallback() vetoMDIClosing()
----------------------------
revision 1.4
date: 2003/01/16 21:08:40;  author: dwallwo;  state: Exp;  lines: +23 -7
added setLock(), removeLock()
----------------------------
revision 1.3
date: 2003/01/14 16:30:24;  author: dwallwo;  state: Exp;  lines: +69 -27
modified namespace.bbj to use new syntax
added adiWithNamespaces.bbj
----------------------------
revision 1.2
```

Standard Output

Status: Succeeded

Figure 7 shows the **`namespace.bbj`** file open in the IDE Source Editor, where CVS has marked the area of the conflict with the lesser-than, equals, and greater-than symbols. Version 1.7 uses bhipple's code on line 18, which conflicts with the attempted change on line 16. It may be a good idea at this point to contact bhipple to determine which line is correct before editing the file and trying to commit once again.

Figure 7. CVS conflict in the IDE Source Editor.

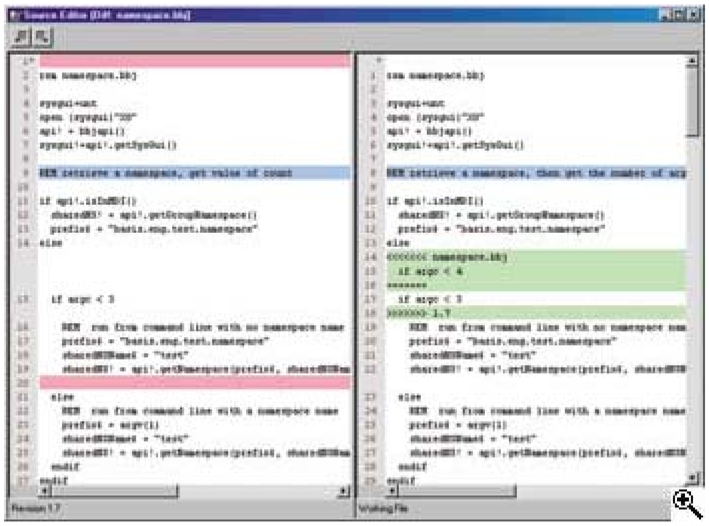


Source Editor [namespace.bbj]

```
 1 rem namespace.bbj
 2
 3 sysgui=unt
 4 open (sysgui)"X0"
 5 api! = bbjapi()
 6 sysgui!=api!.getSysGui()
 7
 8 REM retrieve a namespace, then get the number of arguments
 9
10 if api!.isInMDI()
11   sharedNS! = api!.getGroupNamespace()
12   prefix$ = "basis.eng.test.namespace"
13 else
14 <<<<<<< namespace.bbj
15   if argc < 4
16 =======
17   if argc < 3
18 >>>>>>> 1.7
19     REM  run from command line with no namespace name
20     prefix$ = "basis.eng.test.namespace"
21     sharedNSName$ = "test"
22     sharedNS! = api!.getNamespace(prefix$, sharedNSName$, 1)
23   else
24     REM  run from command line with a namespace name
25     prefix$ = argv(1)
```

One of the best and most useful features of the IDE's CVS client is its ability to graphically display the differences between two different versions of the same file, as shown in **Figure 8**. Here we see the differences between bhipple's version 1.7 of **`namespace.bbj`** and our working copy, containing the conflict. Red highlighting indicates lines present in one version but not the other, while blue highlighting shows the other differences. Green highlighting points out the conflict requiring resolution before rechecking in the file into the repository.

Figure 8. CVS graphical display of differences between disparate versions of the same file.



```
rem namespace.bbj

sysgui=unt
open (sysgui)"X0"
api! = bbjapi()
sysgui!=api!.getSysGui()

REM retrieve a namespace, get value of count

if api!.isInMDI()
  sharedNS! = api!.getGroupNamespace()
  prefix$ = "basis.eng.test.namespace"
else




  if argc < 3

    REM  run from command line with no namespace name
    prefix$ = "basis.eng.test.namespace"
    sharedNSName$ = "test"
    sharedNS! = api!.getNamespace(prefix$, sharedNSNa

  else
    REM  run from command line with a namespace name
    prefix$ = argv(1)
    sharedNSName$ = "test"
    sharedNS! = api!.getNamespace(prefix$, sharedNSNa
  endif
endif
```

```
rem namespace.bbj

sysgui=unt
open (sysgui)"X0"
api! = bbjapi()
sysgui!=api!.getSysGui()

REM retrieve a namespace, then get the number of arg

if api!.isInMDI()
  sharedNS! = api!.getGroupNamespace()
  prefix$ = "basis.eng.test.namespace"
else
<<<<<<< namespace.bbj
  if argc < 4
=======
  if argc < 3
>>>>>>> 1.7
    REM  run from command line with no namespace nam
    prefix$ = "basis.eng.test.namespace"
    sharedNSName$ = "test"
    sharedNS! = api!.getNamespace(prefix$, sharedNS

  else
    REM  run from command line with a namespace name
    prefix$ = argv(1)
    sharedNSName$ = "test"
    sharedNS! = api!.getNamespace(prefix$, sharedNS
  endif
endif
```

All the operations and views shown in **Figures 5 - 8** use the IDE's built-in graphical version control interface, which hides the complexity of the command-line interface and allows software developers to see the status of all their files. Once a development team gets a taste of the power and simplicity offered by a version control system integrated with the IDE, they will wonder how they ever managed without it.

The BBj 3.01 IDE allows Business BASIC Developers to create and maintain applications more easily and efficiently. With the BASIS IDE, developers can integrate a software version control system into their development work. The IDE makes it possible to write code and debug problems faster than ever before. A versioning control system provides more control over development projects by allowing the management of different revisions of project code. Using a VCS system saves time by eliminating duplication of effort, enabling software developers to work more efficiently while producing better quality products. BASIS is committed to providing the best Business BASIC software development tools on the market, and BASIS's goal is to create an environment so powerful, so flexible, and so compelling that nothing else will suffice.