Team Development Using JBuilder®
Refer to the file deploy.html located in the redist directory of your JBuilder product for a complete list of files that you can distribute in accordance with the JBuilder License Statement and Limited Warranty.

Borland Software Corporation may have patents and/or pending patent applications covering subject matter in this document. Please refer to the product CD or the About dialog box for the list of applicable patents. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1997–2002 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. All other marks are the property of their respective owners.

For third-party conditions and disclaimers, see the Release Notes on your JBuilder product CD.

Printed in the U.S.A.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1-1</td>
</tr>
<tr>
<td></td>
<td>Using version control systems</td>
<td>1-2</td>
</tr>
<tr>
<td></td>
<td>Documentation conventions</td>
<td>1-3</td>
</tr>
<tr>
<td></td>
<td>Developer support and resources</td>
<td>1-4</td>
</tr>
<tr>
<td></td>
<td>Contacting Borland Technical Support</td>
<td>1-4</td>
</tr>
<tr>
<td></td>
<td>Online resources</td>
<td>1-5</td>
</tr>
<tr>
<td></td>
<td>World Wide Web</td>
<td>1-5</td>
</tr>
<tr>
<td></td>
<td>Borland newsgroups</td>
<td>1-5</td>
</tr>
<tr>
<td></td>
<td>Usenet newsgroups</td>
<td>1-6</td>
</tr>
<tr>
<td></td>
<td>Reporting bugs</td>
<td>1-6</td>
</tr>
<tr>
<td>2</td>
<td>CVS in JBuilder</td>
<td>2-1</td>
</tr>
<tr>
<td></td>
<td>CVS glossary</td>
<td>2-2</td>
</tr>
<tr>
<td>3</td>
<td>Working on an existing project in CVS</td>
<td>3-1</td>
</tr>
<tr>
<td></td>
<td>Checking out an existing project</td>
<td>3-2</td>
</tr>
<tr>
<td></td>
<td>Posting changes</td>
<td>3-4</td>
</tr>
<tr>
<td></td>
<td>Changes in a single file</td>
<td>3-5</td>
</tr>
<tr>
<td></td>
<td>Changes in the entire project</td>
<td>3-5</td>
</tr>
<tr>
<td></td>
<td>Browsing the project’s CVS status</td>
<td>3-6</td>
</tr>
<tr>
<td></td>
<td>Choosing files to include</td>
<td>3-6</td>
</tr>
<tr>
<td></td>
<td>Committing the project</td>
<td>3-7</td>
</tr>
<tr>
<td></td>
<td>Action options</td>
<td>3-9</td>
</tr>
<tr>
<td></td>
<td>Summary comments</td>
<td>3-10</td>
</tr>
<tr>
<td></td>
<td>Synchronizing with changes in the repository</td>
<td>3-11</td>
</tr>
<tr>
<td></td>
<td>Updating a single file</td>
<td>3-11</td>
</tr>
<tr>
<td></td>
<td>Reconciling CVS merge conflicts</td>
<td>3-11</td>
</tr>
<tr>
<td></td>
<td>Reconciling conflicts automatically</td>
<td>3-12</td>
</tr>
<tr>
<td></td>
<td>Reconciling conflicts manually</td>
<td>3-14</td>
</tr>
<tr>
<td></td>
<td>Updating the project</td>
<td>3-15</td>
</tr>
<tr>
<td></td>
<td>Synchronizing the project file</td>
<td>3-15</td>
</tr>
<tr>
<td></td>
<td>Adding files</td>
<td>3-16</td>
</tr>
<tr>
<td></td>
<td>Removing files</td>
<td>3-17</td>
</tr>
<tr>
<td></td>
<td>Creating a version label</td>
<td>3-17</td>
</tr>
<tr>
<td></td>
<td>Tracking file status in CVS</td>
<td>3-18</td>
</tr>
<tr>
<td></td>
<td>Checking a file’s CVS status</td>
<td>3-18</td>
</tr>
<tr>
<td></td>
<td>File access notification</td>
<td>3-18</td>
</tr>
<tr>
<td></td>
<td>CVS Watches</td>
<td>3-18</td>
</tr>
<tr>
<td>4</td>
<td>Working on a new project in CVS</td>
<td>4-1</td>
</tr>
<tr>
<td></td>
<td>Placing a new project into CVS</td>
<td>4-1</td>
</tr>
<tr>
<td></td>
<td>Checking in a project for the first time</td>
<td>4-2</td>
</tr>
<tr>
<td></td>
<td>Creating a local repository</td>
<td>4-4</td>
</tr>
<tr>
<td>5</td>
<td>Programmer’s guide to the CVS integration</td>
<td>5-1</td>
</tr>
<tr>
<td></td>
<td>Guide to the CVS integration</td>
<td>5-1</td>
</tr>
<tr>
<td></td>
<td>Using CVS as your version control system</td>
<td>5-2</td>
</tr>
<tr>
<td></td>
<td>Selecting CVS</td>
<td>5-2</td>
</tr>
<tr>
<td></td>
<td>Configuring the repository connection</td>
<td>5-2</td>
</tr>
<tr>
<td></td>
<td>Creating a local repository</td>
<td>5-3</td>
</tr>
<tr>
<td></td>
<td>Getting material out of the repository</td>
<td>5-3</td>
</tr>
<tr>
<td></td>
<td>Pulling a project</td>
<td>5-3</td>
</tr>
<tr>
<td></td>
<td>Pulling a project file</td>
<td>5-4</td>
</tr>
<tr>
<td></td>
<td>Checking out files</td>
<td>5-4</td>
</tr>
<tr>
<td></td>
<td>Updating files</td>
<td>5-4</td>
</tr>
<tr>
<td></td>
<td>Removing files</td>
<td>5-5</td>
</tr>
<tr>
<td></td>
<td>Getting material into the repository</td>
<td>5-5</td>
</tr>
<tr>
<td></td>
<td>Checking in a project</td>
<td>5-5</td>
</tr>
<tr>
<td></td>
<td>Adding new files</td>
<td>5-6</td>
</tr>
<tr>
<td></td>
<td>Committing changes</td>
<td>5-6</td>
</tr>
<tr>
<td></td>
<td>Posting a project file</td>
<td>5-6</td>
</tr>
<tr>
<td></td>
<td>Managing the module</td>
<td>5-7</td>
</tr>
<tr>
<td></td>
<td>Version labeling</td>
<td>5-7</td>
</tr>
<tr>
<td></td>
<td>Using watches</td>
<td>5-7</td>
</tr>
<tr>
<td></td>
<td>CVS reference</td>
<td>5-7</td>
</tr>
<tr>
<td></td>
<td>Handling binary files in CVS</td>
<td>5-7</td>
</tr>
<tr>
<td></td>
<td>Checking and setting user environment variables</td>
<td>5-8</td>
</tr>
<tr>
<td></td>
<td>Linux and Solaris</td>
<td>5-8</td>
</tr>
<tr>
<td></td>
<td>Windows XP</td>
<td>5-8</td>
</tr>
<tr>
<td></td>
<td>Windows NT</td>
<td>5-9</td>
</tr>
<tr>
<td></td>
<td>Windows 2000</td>
<td>5-9</td>
</tr>
<tr>
<td></td>
<td>Using CVS with SSH</td>
<td>5-10</td>
</tr>
<tr>
<td></td>
<td>Configuring the SSH connection in JBuilder</td>
<td>5-10</td>
</tr>
<tr>
<td>6</td>
<td>Visual SourceSafe in JBuilder</td>
<td>6-1</td>
</tr>
<tr>
<td></td>
<td>Configuring the connection</td>
<td>6-1</td>
</tr>
</tbody>
</table>
Chapter 12
Working on a new project in ClearCase 12-1
Placing a new project into ClearCase 12-1
Working with views and VOBs 12-2

Chapter 13
Version control reference 13-1
General revision management resources 13-2
Resources for supported tools 13-2
CVS 13-2
Rational ClearCase 13-2
Visual SourceSafe 13-2

Chapter 14
Tutorial: CVS in JBuilder 14-1
Step 1: Setting up 14-2
Step 2: Creating a repository and creating a module 14-3
Step 3: Adding files and committing file changes 14-6
  Adding files to your JBuilder project 14-6
  Adding the files to the CVS repository 14-7
  Modifying files and committing the changes 14-9
Step 4: Checking out an existing module 14-10
  Checking out a module from the repository 14-10
  Modifying a file in the checked out project 14-12
  Removing a file from the checked out project 14-12
  Committing the changes to the repository 14-14
  Step 5: Updating a project 14-14
  Step 6: Simulating multiple users 14-16
    Modify and commit files as User Two 14-16
    Modify and save (not commit) Frame1.java as User One 14-19
    Modify and commit Frame1.java as User Two 14-19
Step 7: Using the Status Browser 14-19
Step 8: Using the Commit Browser 14-22
  Using the Commit Browser to manage files in the version control system 14-22
  Viewing version control status with the Commit Browser 14-22
  Updating the workspace with the Commit Browser 14-23
  Locating and resolving merge conflicts with the Commit Browser 14-24
  Entering comments and committing changes with the Commit Browser 14-26

Index I-1
Team development is safe development when you use effective version control. Version control prevents accidental loss of information while it enables effective version tracking and revision. The two most important reasons to use a version control system are,

- Version control systems make it possible for a team of developers to work on the same set of files without overwriting each other’s changes.

- Version control systems provide logs and version tracking information so that anyone with suitable access can find when any change was made. If you need to backtrack or refer to prior versions of files for any reason, version control makes it feasible.

Many version control systems also provide features for branching and version labeling. Branching allows you to maintain multiple development tracks for the same product, and version labeling allows you to take a snapshot of the entire file set at any stage of development.

JBuilder provides revision handling features in the history view that allow you to maintain file revisions with or without an integrated version control system. JBuilder SE and Enterprise editions provide interfaces with Concurrent Versions System (CVS), Visual SourceSafe, and Rational ClearCase that allow you to perform many version control tasks right from JBuilder. JBuilder’s integration with these version control systems is completely context-sensitive. JBuilder’s Version Control OpenTool and open, extensible architecture allow you to integrate other version control systems as well.
Using version control systems

Version control systems (VCSs) provide ways of storing a complete record of file revisions while allowing developers to continue revising. Generically, we can call the storage area the repository and the working area the workspace. Although different VCSs are structured uniquely, and each VCS has its own terminology, for general conceptual information presented in this guide, the terms repository and workspace will be used as defined here.

Normally, the repository stores the current master version of each file and maintains a record of all the changes that have been made to each file. The workspace has the version of each file that an individual user has most recently updated and modified. Some VCSs let only one user at a time use a file (pessimistic or locking model), some let multiple users use the same file simultaneously (optimistic or concurrent development model), and some blend these approaches.

When revisions are posted from the workspace to the repository, the version control system stores the changed parts of the master copy. The rest of the file is unchanged. The part that was revised is stored in the repository, along with information about where it came from and when it was changed.

To benefit from version control, users must,

- Retrieve files or groups of files from the repository and place them in their individual workspaces
- Synchronize their local versions of the files with the repository versions
- Place the revisions back in the repository when they have finished working on the files

When developers use this process, all changes are available to every developer who has access to that repository.
Documentation conventions

The Borland documentation for JBuilder uses the typefaces and symbols described in the following table to indicate special text.

Table 1.1  Typeface and symbol conventions

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monospaced type</td>
<td>Monospaced type represents the following:</td>
</tr>
<tr>
<td></td>
<td>• text as it appears onscreen</td>
</tr>
<tr>
<td></td>
<td>• anything you must type, such as “Type Hello World in the Title field of the Application wizard.”</td>
</tr>
<tr>
<td></td>
<td>• file names</td>
</tr>
<tr>
<td></td>
<td>• path names</td>
</tr>
<tr>
<td></td>
<td>• directory and folder names</td>
</tr>
<tr>
<td></td>
<td>• commands, such as <code>SET PATH</code></td>
</tr>
<tr>
<td></td>
<td>• Java code</td>
</tr>
<tr>
<td></td>
<td>• Java data types, such as boolean, int, and long.</td>
</tr>
<tr>
<td></td>
<td>• Java identifiers, such as names of variables, classes, package names, interfaces, components, properties, methods, and events</td>
</tr>
<tr>
<td></td>
<td>• argument names</td>
</tr>
<tr>
<td></td>
<td>• field names</td>
</tr>
<tr>
<td></td>
<td>• Java keywords, such as <code>void</code> and <code>static</code></td>
</tr>
<tr>
<td>Bold</td>
<td>Bold is used for Java tools, <code>bmj</code> (Borland Make for Java), <code>bcj</code> (Borland Compiler for Java), and compiler options. For example: <code>javac</code>, <code>bmj</code>, <code>-classpath</code>.</td>
</tr>
<tr>
<td>Italics</td>
<td>Italicized words are used for new terms being defined, for book titles, and occasionally for emphasis.</td>
</tr>
<tr>
<td>Keycaps</td>
<td>This typeface indicates a key on your keyboard, such as “Press <code>Esc</code> to exit a menu.”</td>
</tr>
<tr>
<td>[ ]</td>
<td>Square brackets in text or syntax listings enclose optional items. Do not type the brackets.</td>
</tr>
<tr>
<td>&lt; &gt;</td>
<td>Angle brackets are used to indicate variables in directory paths, command options, and code samples.</td>
</tr>
<tr>
<td></td>
<td>For example, <code>&lt;filename&gt;</code> may be used to indicate where you need to supply a file name (including file extension), and <code>&lt;username&gt;</code> typically indicates that you must provide your user name.</td>
</tr>
<tr>
<td></td>
<td>When replacing variables in directory paths, command options, and code samples, replace the entire variable, including the angle brackets (<code>&lt; &gt;</code>). For example, you would replace <code>&lt;filename&gt;</code> with the name of a file, such as <code>employee.jds</code>, and omit the angle brackets.</td>
</tr>
<tr>
<td></td>
<td>Note: Angle brackets are used in HTML, XML, JSP, and other tag-based files to demarcate document elements, such as <code>&lt;font color=red&gt;</code> and <code>&lt;ejb-jar&gt;</code>. The following convention describes how variable strings are specified within code samples that are already using angle brackets for delimiters.</td>
</tr>
</tbody>
</table>
Developer support and resources

Borland provides a variety of support options and information resources to help developers get the most out of their Borland products. These options include a range of Borland Technical Support programs, as well as free services on the Internet, where you can search our extensive information base and connect with other users of Borland products.

Contacting Borland Technical Support

Borland offers several support programs for customers and prospective customers. You can choose from several categories of support, ranging from free support on installation of the Borland product to fee-based consultant-level support and extensive assistance.

Table 1.1  Typeface and symbol conventions (continued)

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Italics, serif</em></td>
<td>This formatting is used to indicate variable strings within code samples that are already using angle brackets as delimiters. For example, <code>&lt;url=&quot;jdbc:borland:jbuilder\samples\guestbook.jds&quot;&gt;</code></td>
</tr>
<tr>
<td>...</td>
<td>In code examples, an ellipsis (…) indicates code that has been omitted from the example to save space and improve clarity. On a button, an ellipsis indicates that the button links to a selection dialog box.</td>
</tr>
</tbody>
</table>

JBuilder is available on multiple platforms. See the following table for a description of platform conventions used in the documentation.

Table 1.2  Platform conventions

<table>
<thead>
<tr>
<th>Item</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paths</td>
<td>Directory paths in the documentation are indicated with a forward slash (/). For Windows platforms, use a backslash ().</td>
</tr>
<tr>
<td>Home directory</td>
<td>The location of the standard home directory varies by platform and is indicated with a variable, <code>&lt;home&gt;</code>.</td>
</tr>
<tr>
<td></td>
<td>• For UNIX and Linux, the home directory can vary. For example, it could be <code>/user/&lt;username&gt;</code> or <code>/home/&lt;username&gt;</code></td>
</tr>
<tr>
<td></td>
<td>• For Windows NT, the home directory is <code>C:\Winnt\Profiles\&lt;username&gt;</code></td>
</tr>
<tr>
<td>Screen shots</td>
<td>Screen shots reflect the Metal Look &amp; Feel on various platforms.</td>
</tr>
</tbody>
</table>
For more information about Borland’s developer support services, see our web site at http://www.borland.com/devsupport/, call Borland Assist at (800) 523-7070, or contact our Sales Department at (831) 431-1064.

When contacting support, be prepared to provide complete information about your environment, the version of the product you are using, and a detailed description of the problem.

For support on third-party tools or documentation, contact the vendor of the tool.

Online resources

You can get information from any of these online sources:

**World Wide Web**  http://www.borland.com/

**FTP**  ftp://ftp.borland.com/

Technical documents available by anonymous ftp.

**Listserv**  To subscribe to electronic newsletters, use the online form at:
http://info.borland.com/contact/listserv.html

or, for Borland’s international listserv,
http://info.borland.com/contact/intlist.html

World Wide Web

Check www.borland.com/jbuilder regularly. This is where the Java Products Development Team posts white papers, competitive analyses, answers to frequently asked questions, sample applications, updated software, updated documentation, and information about new and existing products.

You may want to check these URLs in particular:

- http://www.borland.com/jbuilder/ (updated software and other files)
- http://www.borland.com/techpubs/jbuilder/ (updated documentation and other files)
- http://community.borland.com/ (contains our web-based news magazine for developers)

Borland newsgroups

You can register JBuilder and participate in many threaded discussion groups devoted to JBuilder. The Borland newsgroups provide a means for the global community of Borland customers to exchange tips and techniques about Borland products and related tools and technologies.
Developer support and resources

You can find user-supported newsgroups for JBuilder and other Borland products at http://www.borland.com/newsgroups/.

Usenet newsgroups

The following Usenet groups are devoted to Java and related programming issues:

- news:comp.lang.java.advocacy
- news:comp.lang.java.announce
- news:comp.lang.java.beans
- news:comp.lang.java.databases
- news:comp.lang.java.gui
- news:comp.lang.java.help
- news:comp.lang.java.machine
- news:comp.lang.java.programmer
- news:comp.lang.java.security
- news:comp.lang.java.softwaretools

Note: These newsgroups are maintained by users and are not official Borland sites.

Reporting bugs

If you find what you think may be a bug in the software, please report it in the Support Programs page at http://www.borland.com/devsupport/namerica/. Click the “Reporting Defects” link to bring up the Entry Form.

When you report a bug, please include all the steps needed to reproduce the bug, including any special environmental settings you used and other programs you were using with JBuilder. Please be specific about the expected behavior versus what actually happened.

If you have comments (compliments, suggestions, or issues) for the JBuilder documentation team, you may email jgpubs@borland.com. This is for documentation issues only. Please note that you must address support issues to developer support.

JBuilder is made by developers for developers. We really value your input.
CVS support is a feature of JBuilder SE and Enterprise editions. JBuilder SE and Enterprise editions integrate Concurrent Versions System (CVS), a widely used open-source version control system. JBuilder has a responsive interface that provides context-sensitive access to the most common CVS commands from within the AppBrowser.

When you install JBuilder SE or Enterprise edition, CVS is automatically installed into the `<jbuilder>/bin` directory.

You can create modules, check out projects, add or remove files, check the CVS status of any file or project, commit changes from within the JBuilder interface, use watches and CVS edit, create version labels and access branches, all from within the JBuilder IDE. If you’re unsure what these or other CVS terms mean, please check the “CVS glossary” on page 2-2.

CVS categorizes all files as either text-based or binary. Code is normally treated as text-based, and image files and user-defined binary types are treated as binary. For more information on how CVS handles binary files, please see “Handling binary files in CVS” on page 5-7.

**Note**

CVS configuration information is automatically requested when you check out or check in a project.

When a project is in CVS, choose Team | Configure CVS to view its connection configuration. Once the connection has been established, this display becomes read-only. This configuration can also be viewed in your project file settings.
These words have specific meanings in CVS:

**Add**
Adds a file to the module.

**Checkout**
Pulls a module from the repository to your workspace. When using CVS, this should only be done the first time. After that, files can be synchronized by updating them.

**Commit**
Applies changes from your workspace to the repository. You must commit file changes, file additions, and file deletions to the version control system to make the revised file available to other users.

**Merge**
CVS and some other systems don’t necessarily lock files when they are being used. To preserve all changes, these systems use the merge command: it combines changes from the repository with changes in the workspace. Saved workspace changes are not overwritten, and textual conflicts are preserved and flagged to be reconciled by the user. In CVS, an update includes a merge.

**Module**
A container for a group of associated files that are stored together in the repository for better file management and user convenience.

**Project**
Can have any of three meanings: the files and settings that make up one body of work, the organizational file used in JBuilder that manages the list of those files and settings, or as a synonym for “module”.

**Remove**
Removes a file from the repository. The file must first be deleted from your workspace.

**Repository**
Where the modules and revision records that you have put into version control are kept. This may be on your local machine or on a remote server.

**Update**
Retrieves changes from the repository and applies them to your workspace.

**Workspace**
The area you affect directly, and which you must maintain yourself. When you make changes to a file, you make them and save them in the workspace first.
A JBuilder project must be under CVS control before CVS menu commands become available. This means you must either pull a project from CVS or place an existing project into CVS.

You’ll access CVS most often by pulling an existing project from the CVS repository.

To work on an existing project, you must first check it out into your workspace. Checking out a project into your workspace in JBuilder does three important things:

- Mirrors the current repository version of the project into your workspace, so that you have the most current version at the time of checkout.
- Notifies CVS that you have it. This engages version control management mechanisms, so that records of changes can be kept, new versions can be generated as necessary, and conflicts can be flagged.
- Notifies JBuilder that you’re using the project under version control. This engages JBuilder’s support features, activating the Team development features, allowing you to use the history pane to greater effect, and providing conflict management assistance.

**Note** When working under version control, it’s important to use it appropriately. Consistent version control is easy version control, but, if used irregularly or inconsistently, it becomes more difficult to use.
Checking out an existing project

The quickest way to check out a project that’s already in the repository is to use the Pull Project From CVS wizard available in the object gallery:

1. Choose File | New to open the object gallery.
2. Click the Project tab and select Pull Project From CVS.
3. Either click OK, double-click the Pull Project From CVS icon, or press *Enter* to launch the Pull Project From CVS wizard.
4. Type or select an empty target directory.
   JBuilder offers an empty *untitled* directory by default. Change the directory name to something meaningful, as long as that directory either doesn’t exist yet or is completely empty.
5. Click Next to proceed to Step 2.

Under Connection Type, *Local* connects to a local repository, *PServer* connects to a repository on a server that’s normally password-protected, and *Ext* connects to a repository on a secure server. If you select PServer, you have the option of specifying a port number for connecting to the remote CVS repository host.

**Note**

This special port number for PServer connections is required only if the CVS server has been configured to use a port number different from the default, 2401.

The *CVSROOT* path appears at the bottom of the wizard. It reflects the information entered in this step. If you’re unsure what you should enter and you already have a CVS account on an existing server, see your CVS administrator.
Checking out an existing project

**Note**  
JBuilder maintains drop-down lists of prior connections for you to choose from. These lists are empty when you use the wizard for the first time, but after that you can select previous entries from the lists instead of typing them in.

6 Make your entries and click Next to proceed to Step 3.

7 Type in or choose the repository path.  
If you’re using a local repository, you can click the ellipsis (...) button to browse to it instead.

8 Type in or choose a module name.  
Click the Scan button next to the Module Name combo box to scan for existing modules defined in the CVSROOT/Modules file. If the CVSROOT/Modules file exists, and contains a list of modules, the Modules Name combo box will be populated with the module names, in alphabetical order.

9 Type in or choose a branch name.  
If you want to work on a branch other than the main branch, type in or choose the branch from those that are available in the drop-down list. The default is the main trunk (<main>). Click the Scan button next to the Branch combo box to populate the Branch combo box with a list of branches for the selected module. The <main> branch is listed at the top, followed by all other branches in alphabetical order.

10 Optionally, you can check Autosave Files Before CVS Operations or Show Console Messages.  
Autosave Files Before CVS Operations will save you a version control step. If this box is unchecked and you execute a CVS command from JBuilder without saving the changes first, you will be prompted to save the files then. This option is checked by default.
Posting changes

Show Console Messages enables the display of stdout output from CVS in the message pane. This allows you to view the exact command-line feedback from CVS on each operation. This option is off by default.

11 Click Finish to complete the wizard.

The Checking Out CVS Project dialog box appears and reports on the progress of the checkout.

12 Click OK to close the dialog box.

When JBuilder pulls a project from CVS, a directory view node is added at the root of the project in the project pane. The directory view shows the entire project directory tree with version control subdirectories hidden.

This is the second way to initiate this wizard:

1 Open or create a project that’s not under version control.

Note

An existing project will not be overwritten. Opening a project simply gives you access to the Team menu.

2 Choose Team | Select Project VCS.

The Select Project VCS dialog box appears.

3 Select CVS.

4 Click OK or press Enter to close the dialog box.

5 Choose Team | Pull Project From CVS.

The Pull Project From CVS wizard appears.

Posting changes

When you post changes, you apply your altered file to the repository. The repository records where the changes were made and generates a new master copy of the file. When other users retrieve that file from the repository, they will get the version that includes the changes you made. In CVS, posting changes is called either committing or checking in changes; these terms are used interchangeably.
Changes in a single file

When you have made changes to a file in your workspace, you can use the Commit command to update the repository with a new version of the file.

To post changes for the active file, use the Team menu:

1. Choose Team | Commit “<filename>”.
   The CVS Commit dialog box appears.
2. Enter a comment to describe the changes you made to the file and click OK.
3. When the Commit command has successfully completed, click OK to close the dialog box.

Note: By default, JBuilder displays dialog boxes to confirm the success or failure of version control system (VCS) operations. To configure these confirmation dialog boxes to close automatically after successful VCS operations, check Close VCS Dialogs Automatically After Successful Operation in the IDE Options dialog box (Tools | IDE Options).

For any file available in the project pane, use the context menu:

1. Right-click the file in the project pane.
2. Choose CVS | Commit “<filename>”.
   The CVS Commit dialog box appears.
3. Enter a comment to describe the changes you made to the file and click OK.
4. When the Commit command has successfully completed, click OK to close the dialog box.

This command checks in a single changed file to the repository. This keeps the repository version of the file current and maintains a good revision history.

Update your workspace before you commit file changes. Updating significantly reduces the risk of merge conflicts, and, where they do occur, engages JBuilder’s merge assistance mechanisms.

Changes in the entire project

It’s not always feasible to commit changes to each file as you go. You may need to test files to make sure your changes work or you might even forget once in awhile. You may not exactly remember which ones you changed and how. JBuilder lets you browse the status of all the files in your project using the Status Browser, and commit as many files as necessary using the Commit Browser.

Note: The current project file is maintained separately.
Browsing the project’s CVS status

Choose Team | Status Browser to see all the files in your project and the CVS status of each one. The Status Browser has two pages: the Changes page and the File Include Lists page.

In the Changes page, the project’s directory tree is displayed in the left pane and the list of files in the right. Select the Full List node to view a list of all the files in the project. Select a directory node to see a list of the files in that directory.

The version control status of each file is noted in the Status column of the list. Select a file in the file list. The tabbed pane below the file list allows you to view the source code of the selected file in any way that’s pertinent. If you choose a file that has been changed locally, then the Workspace Source, Repository Source, and Workspace Diff tabs become available. If you choose a file that has been changed on the repository, then the Workspace Source, Repository Source, Repository Diff, and Complete Diff tabs become available.

Table 3.1  Changes page source views

<table>
<thead>
<tr>
<th>Source view</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workspace Source</td>
<td>This file’s source code from the current workspace version.</td>
</tr>
<tr>
<td>Repository Source</td>
<td>This file’s source code from the current repository version.</td>
</tr>
<tr>
<td>Workspace Diff</td>
<td>This file’s most recent changes in your workspace.</td>
</tr>
<tr>
<td>Repository Diff</td>
<td>This file’s most recent changes in the repository.</td>
</tr>
<tr>
<td>Complete Diff</td>
<td>Differences between the current version of this file in the repository and the current version in your workspace.</td>
</tr>
</tbody>
</table>

Choosing files to include

The File Include Lists determine which files are under version control. There are two pages: Team Include List and Personal Include List.
The Team Include List files are tracked by the `<projectname>.jpx` file. The information you enter here is stored in that file as well, because this is a team-wide project setting: the files that are checked here are the files that everyone needs to be able to use.

The backup files (bak directory) and the `<projectname>.jpx.local` file are normally excluded (unchecked). Check your company’s policy about what files should be included and what should be excluded from a team project checkin.

**Caution** The shared .jpx file must be included (checked in the Team Include List) to maintain version control for the project with CVS in JBuilder.

The Personal Include List files are tracked by the `<projectname>.jpx.local` file (where the information in this table is stored.) This list is entirely for your convenience. Since you won’t necessarily be working on every file in the project, you don’t necessarily want to look at them whenever you do a checkin. This simple chart illustrates the concept of available files as compared to the files needed by an individual developer:

```
EntireProject
  UFiles
    UFile1
    UFile2
  DataFiles
    DataFile1
    DataFile2
    DataFile3
    DataFile4
  Beans
    Bean1
    Bean2
    Bean3
```

The Personal Include List page lets you keep in view only the files that you need. The rest stay hidden until you want to see them again.

**Committing the project**

The Commit Browser provides status messages and context-sensitive default actions. It also provides alternative action options in the form of combo boxes in the Action column. Click a file’s cell in the Action column to view alternative actions. These vary depending on the status of the file. The default option varies depending on how the file achieved that status.

The Commit Browser also allows you to determine which files will be included under CVS control and which files will remain strictly local. It
has three pages: Commits, Summary Comment, and File Include Lists. It opens to the Commits page by default.

An expandable directory tree view is in the left pane and the list of files is in the right. Use them like this,

- To view a list of all the files in the project, select the Full List node in the directory tree
- To display subdirectories in the directory tree, simply expand the module or parent directories
- To display files in the file list for an individual directory, select the directory node in the directory tree
- To display or enter a comment for an individual file, select the file from the file list, and select the Individual Comment pane in the bottom half of the Commit Browser.

The tabbed pane also displays the selected file’s source in five ways:

<table>
<thead>
<tr>
<th>Tab</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workspace Source</td>
<td>This file’s source code from the current workspace version.</td>
</tr>
<tr>
<td>Repository Source</td>
<td>This file’s source code from the latest repository version.</td>
</tr>
<tr>
<td>Workspace Diff</td>
<td>This file’s most recent changes in your workspace.</td>
</tr>
<tr>
<td>Repository Diff</td>
<td>This file’s most recent changes in the repository.</td>
</tr>
<tr>
<td>Complete Diff</td>
<td>The differences between the current local version of this file and the latest repository version.</td>
</tr>
</tbody>
</table>
Action options

JBuilder chooses default options to place in the Action column based on the file’s status as reflected in the Status column. For changes made within JBuilder, the default actions are chosen not only according to the file’s status but how it reached that status. For instance, if a file isn’t in the workspace, it might be because it was removed from the project or because it has not yet been checked out. JBuilder senses the reason that it’s not in the workspace and chooses the most likely option to list as the default: Remove From Repository or Check Out. Additions and removals made from this browser are automatically committed.

If you want to commit some but not all changed files in the project, select No Action as the option for the files you don’t want to commit. You can also auto-select No Action for all of the files in a directory by right-clicking the directory in the node tree view and choosing Perform No Action For All In Directory from the context menu.

For the default option to be the most intelligent, file additions and deletions must be performed from within JBuilder. Then JBuilder can perceive the reason for the changed status of a file and select the most appropriate default option.

Options include:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>Commit the change to the CVS repository.</td>
</tr>
<tr>
<td>Add</td>
<td>Add and commit this file so it’s stored in the repository.</td>
</tr>
<tr>
<td>Delete</td>
<td>This file is not in the repository; this removes it from your workspace.</td>
</tr>
<tr>
<td>Get</td>
<td>This file has been added to the repository; this checks it out to your workspace.</td>
</tr>
<tr>
<td>Revert</td>
<td>Update the workspace with the latest repository version of this file, discarding all changes made since your last update.</td>
</tr>
<tr>
<td>Update</td>
<td>This file has changed in the repository; this updates it in your workspace.</td>
</tr>
<tr>
<td>No Action</td>
<td>This changed file will not be touched by a CVS operation of any kind. It will be exactly as you left it before you invoked the Commit Browser.</td>
</tr>
</tbody>
</table>

If committing a file from the Commit Browser creates a merge conflict, the conflicts are clearly marked with “<<<<<<”, “=======”, and “>>>>>>, and JBuilder’s merge conflict handling mechanisms is engaged. For example,
the following code shows a conflict between to comments in a file named Frame1.java:

```java
package thisproject;
// second change
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
<<<<<<< Frame1.java
// fifth change
=======
// conflicts with the fifth change
>>>>>>> 1.5
/**
 * <p>Title: </p>

For more information on these, see “Updating a single file” on page 3-11.

The following table demonstrates a few status notations and associated options.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Listed status</th>
<th>Default option</th>
<th>Alternative options</th>
</tr>
</thead>
<tbody>
<tr>
<td>File changed in the workspace</td>
<td>Changed In Workspace</td>
<td>Commit to Repository</td>
<td>• Revert</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No Action</td>
</tr>
<tr>
<td>File added to CVS</td>
<td>Not In Repository</td>
<td>Add to Repository</td>
<td>• Delete Locally</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No Action</td>
</tr>
<tr>
<td>File deleted from CVS</td>
<td>Not In Workspace</td>
<td>If removed from within JBuilder:</td>
<td>• Revert</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove From Repository</td>
<td>• No Action</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If removed from outside JBuilder:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Get From Repository</td>
<td></td>
</tr>
</tbody>
</table>

Summary comments

The Use Summary Comment checkbox at the bottom of the Individual Comment pane allows you to attach the same summary comment to multiple files. This summary comment will be maintained along with the individual comment written for each file. Select the Summary Comment tab to write a summary comment.

The Summary Comment page offers the option of placing the summary comment before the individual comments applied to individual files. This option is on by default. If you deselect this option and select the Use Summary Comment option on the Commits page, then the summary comment will be applied to files that don’t have individual comments.
Synchronizing with changes in the repository

Updating pulls the current repository version into your workspace. You can update individual files or the entire project at once.

**Note**  Project files are updated separately.

### Updating a single file

When you update your workspace with changes from the repository, CVS automatically merges repository changes with your changes. This reduces merge conflicts when you commit your changes back to the repository.

When different changes are made to the same area of text, CVS registers a conflict and will not let the later user commit changes to that file. Conflicts are clearly marked in the file with “<<<<<<<”, “======”, and “>>>>>>”. When JBuilder finds a merge conflict, it displays a conflict message in the message pane. Selecting the conflict message displays the Merge Conflicts page in the history pane, with the conflicting areas in the workspace source and repository source highlighted.

**Note** CVS recognizes only textual conflicts, not logical ones. A textual conflict is a region of overlapping text: different characters written into exactly the same space. A logical conflict is a programmatic event, when incompatible or problematic programming elements are used in the same program. CVS is not designed to handle logical evaluations of programs, only physical assessments of files.

CVS is designed to handle text-based files. It treats all other files as binary. CVS can update or commit binary files, but doesn’t merge them. See “Handling binary files in CVS” on page 5-7.

Update before committing. It reduces merge conflicts. JBuilder prompts you to update before committing when necessary. Updating a file with conflicts gives JBuilder the chance to flag the conflicts and it sets the features that make finding and resolving merge conflicts much easier.

### Reconciling CVS merge conflicts

JBuilder helps avoid merge conflicts by requiring you to update when necessary before committing changes. If merge conflicts do occur, CVS and JBuilder alert you in the following ways:

- JBuilder displays a conflict message in the message pane. Click a merge conflict warning within the message pane to display the Merge Conflicts page in the history pane.

- CVS inserts “<<<<<<<”, “======”, and “>>>>>>” around the conflicting blocks in the file in your workspace. For example, if you modified a
Synchronizing with changes in the repository

comment in the file in your workspace, but someone else modified the
same comment and checked the changes in, the conflict might be
tagged like the following code sample:

`<<<<<< Frame1.java
// Comment modified in file in the workspace
======
// Comment in file checked into repository
>>>>>> 1.3`

This tagging is caught by the compiler, so if you postpone resolving
conflicts in Java files, you can find them again later by compiling the
file and using the message pane.

- JBuilder’s Merge Conflicts page in the history pane displays the
  workspace source side-by-side with the repository source, with the
  conflicting blocks of code or text highlighted. Radio buttons next to the
  highlighted blocks of conflicting code allow you to easily select which
  block of code you want to keep. The preview pane at the bottom of the
  Merge Conflicts page shows what your workspace file will look like
  when you apply the changes.

There are two ways to reconcile CVS merge conflicts in JBuilder:

1. Reconcile them automatically, using the radio buttons in the Merge
   Conflicts page in the history pane to choose the code block to keep. The
   Merge Conflicts page is enabled only when JBuilder detects a conflict.

2. Reconcile them manually, searching for the “<<<<<<”, “======”, and
   “>>>>>>” tagging in the file source, and using the editor to resolve the
   conflicts. Remember to delete the conflict tagging and any extraneous
   text when you’re done.

Choose the way that best addresses the conflict.

Reconciling conflicts automatically

JBuilder’s automatic merge conflict handling mechanisms are suitable for
straightforward conflicts where sequential changes were made to the
same block of text.

To choose automatically between old and new versions of a conflicting
area of text,

1. Go the history pane for the file, and open the Merge Conflicts page.

   The conflicting areas in the workspace source and repository source are
   highlighted in separate panes on the Merge Conflicts page. Radio
   buttons in the gutters of the workspace and repository panes are used
   to select the changes to keep.

   Lines for a conflict that has been selected to be kept are highlighted in
   yellow, and are further indicated by plus (+) signs in the gutter.
Synchronizing with changes in the repository

Changes to be discarded are highlighted in red and have minus (-) signs in the gutter. The preview pane at the bottom of the Merge conflicts page shows what the file source will look like when the changes are applied. By default, the changes in the workspace are selected to be retained.

If there are multiple conflicts, you can click the navigation arrows at the bottom left corner of the page to scroll forward and back between pairs of conflicts.

2 Click the radio button next to the version (workspace or repository) of the block of text that you want to keep.

When you click a radio button, the highlight color changes to yellow, and a plus (+) sign appears in the gutter of the selected line or lines. The preview pane updates to show how the file will look if the changes are applied. The icon in the gutter of the preview pane changes to indicate the source (workspace or repository) of the selected change.

3 Click Apply to apply the change in the editor buffer.

Applying changes updates the editor buffer for the file in the workspace. Until you commit your changes, you can use the undo command (Edit | Undo or Ctrl+z) to restore the conflicts. The undo command is not available from the history pane, so you will need to switch to a different view.

4 Save the file and commit the changes.
Reconciling conflicts manually
If you don’t want to use JBuilder’s automatic merge conflict handling mechanisms, you can edit files manually to resolve conflicts.

To manually reconcile conflicts,

1. **Search for the “<<<<<<<”, “========”, and “>>>>>>>” tagging in the file source.**

   ```java
   package thisproject;
   // second change
   import java.awt.*;
   import java.awt.event.*;
   import javax.swing.*;
   // fifth change
   // conflicts with the fifth change
   >>>>>>> 1.5
   /**
    *  
    *   
    *  
    */
    * <p>Title: </p>
    * <p>Description: </p>
    * <p>Copyright: Copyright (c) 2002</p>
    * <p>Company: </p>
    * @author H. Marx
    * @version 1.0
    */
    
2. **Edit the text as you normally would until the conflict is resolved.**

   ```java
   package thisproject;
   // second change
   import java.awt.*;
   import java.awt.event.*;
   import javax.swing.*;
   // conflicts with the fifth change
   /**
    *  
    *   
    *  
    */
    * <p>Title: </p>
    * <p>Description: </p>
    * <p>Copyright: Copyright (c) 2002</p>
    * <p>Company: </p>
    * @author H. Marx
    * @version 1.0
    */
    
   Be sure to remove the conflict tagging and any extraneous text.

3. **Save the file and commit the changes.**

   **Note** Once conflicts have been tagged, CVS no longer sees them as conflicts.

   **Tip** The compiler will report errors if it finds conflict tags when it compiles. Therefore, if you postpone resolving a set of conflicts, you can compile and then double-click the compiler’s error message to find the conflicts in the editor.
Synchronizing with changes in the repository

The Status Browser dialog box (Team | Status Browser) provides another means of viewing differences between file revisions (difs) and conflicts. Note that this is only a view, not a context for change.

When all merge conflicts have been resolved, commit the changes and update normally.

**Updating the project**

Updates all the files in your workspace with changes in the repository. Any differences between the repository version of a file and your workspace version of that file are automatically merged. Conflicts are flagged.

For more information on handling merges and merge conflicts, see “Reconciling CVS merge conflicts” on page 3-11.

**Synchronizing the project file**

Instead of automatically being committed and updated with the rest of the project, the project file must be committed and updated in a separate process. This allows you to control when and whether global settings get posted to the repository.

**Important**

Complex projects may contain other projects within them. This means that one project may have several project files, at different levels within it. Only the current, top-level .jpx project file for the currently active project is handled separately. Other project files in the project are updated with the rest of the project.

To commit your local version of the project file to the repository, choose Team | Post Current "<project file name>".

To update your workspace with the latest repository version of the project file, choose Team | Pull Latest "<project file name>".

The project file is maintained separately from the rest of the project. There are two reasons for this:

- Changes made to the project file can include path settings and other changes that would affect the work of others who share the same project. Maintaining the project file separately allows you to make changes to files and paths and to test those changes before you need to alter the project file.

- Many people may be working on the same project, but using different files within it. If each of them commits the project file every time they commit the project, it can create difficulties for the other users of the same project.
Adding files

Adding a file to the JBuilder project is separate from adding it to the CVS module. Also, a file that will be added to CVS must be within the project directory before it gets added to the JBuilder project, or else JBuilder will have the wrong path for it.

Therefore, the sequence of actions before a file can be added to CVS is,

1. Put the file in the project directory.
2. Add the file to the JBuilder project.
3. Add the file to the CVS module.

The addition of the file must be committed in a separate step. Keeping the Add and Commit commands separate allows you to work with the addition locally before deciding whether to commit it to the repository.

To add the active file to the CVS module,

1. Choose Team \rightarrow Add File.
2. Enter a comment to describe the addition, according to your usual practices.
3. Click OK in the dialog box to initiate the addition.
4. Click OK to close the dialog box.

To add an inactive file or multiple files to the CVS module,

1. Select the file nodes in the project pane.
   Use the Ctrl key or the Shift key to select multiple nodes.
2. Right-click the selection and choose CVS \rightarrow Add Files from the context menu.
   If you are adding a single file, the format of the menu option will be CVS \rightarrow Add “<filename>”.
3. Enter a comment to describe the additions, according to your usual practices.
4. Click OK in the dialog box to accept the files listed.
5. Click OK to close the dialog box.

**Caution** While files referred to in a JBuilder project can reside in different directory trees, files in a CVS module must reside in a single tree. Therefore, before you add a file to CVS, make sure you have copied or moved it into the project directory in your workspace.

**Tip** If you have trouble adding a file, read the note in the status bar. It will tell you the source of the trouble.
Removing files

Deleting a file from CVS within JBuilder removes the active file from the repository, from your workspace, and from the project.

To remove the active file from the CVS module,

1. Choose Team | Remove File.
2. Enter a comment to describe the removal, according to your usual practices.
3. Click OK in the dialog box to initiate the removal.
4. Click OK to close the dialog box.

To remove an inactive file or multiple files from the CVS module,

1. Select the file nodes in the project pane.
   Use the Ctrl key or the Shift key to select multiple nodes.
2. Right-click the selection and choose CVS | Remove Files from the context menu.
   If you are removing a single file, the format of the menu option will be CVS | Remove “<filename>”.
3. Enter a comment to describe the removals, according to your usual practices.
4. Click OK in the dialog box to accept the files listed.
5. Click OK to close the dialog box.

Creating a version label

Version labels, or tags, allow you to take a snapshot of the entire project at any point in time. Since different files change at different rates, a project of 100 files can contain 100 different current revision numbers. Version labels mark the evolution of the entire project without reference to changes in individual files.

Choose Team | Add Version Label To Project to create a version label for the current project. The Add Version Label To Project dialog box appears.

Name the version label according to your usual practices. Describe the label’s purpose. Click OK when you’re done. JBuilder applies the label to every file in the project that resides in the repository.
Tracking file status in CVS

If other developers are working on the same files as you, it is a good idea to check the status of files to reduce the risk of conflicts.

Checking a file’s CVS status

Displays the current status of the selected file in CVS: whether changes have been made locally or remotely, whether conflicts have been found, and so on.

To check the status of the active file, choose Team | Status For “<filename>”.

To check the status of any single file in the project,
1 Select the file in the project pane.
2 Right-click the selected file, and choose CVS | Status For “<filename>” from the context menu.
3 Click OK to close the dialog box.

File access notification

JBuilder supports `cvs watch`, `cvs edit`, and `cvs unedit` commands.

Normally, you learn that other developers have changed a file when you update it or check its status. CVS is an optimistic version control system, allowing more than one developer to make changes to the same set of files at the same time. It’s possible to notify each other when you’re working on common files with CVS by using the CVS Watch functions.

These commands are available from the Team menu (Team | CVS Watches) and from the CVS context menu in the project pane. Select multiple files from the project pane and use the context menu to put a Watch on or apply CVS Edit to more than one file at a time.

Note The CVS administrator must put a watch on the files before these commands are useful.

CVS Watches

CVS Watches provide a way to notify you when someone else is working on a file that you have checked out. This is especially useful for a large or geographically dispersed group of users who are collaborating on the same set of files.

The watch functions are supported in JBuilder to accommodate environments that use watches. Watch functions, even more than most
version control commands, require cooperation on every user’s part to be effective.

JBuilder supports the CVS commands `cvs watch add`, `cvs watch remove`, `cvs edit`, and `cvs unedit`. The CVS administrator (or someone with appropriate access) must enable watches before they’re useful in JBuilder.

Choose Team | CVS Watches to access the CVS watches commands:

A notification is sent to all users when CVS Edit is used on a file—depending on the server’s configuration, this could be an email, pager call, or log file entry.

JBuilder’s CVS Edit command can be terminated in one of two ways, depending on whether you want to keep the changes you made or not:

1. If you want to keep your changes, commit the file.
   This will remove the CVS Edit, while leaving the Watch on the file.

2. If you changed your mind about changing the file, use CVS UnEdit.
   The CVS UnEdit command will,
   - Cancel all the changes that you made since applying the CVS Edit
   - Remove the CVS Edit
   - Leave the Watch on
   - Generate a log entry

These are menu-driven commands. There are no dialog boxes associated with them. To view a file’s Watch and Edit status, choose Team | Status For “<filename>” with that file active in the content pane, or choose Team | Status Browser and look for the file on the Changes page.
Chapter 4

Working on a new project in CVS

Once JBuilder is notified that you want to put the active project under version control, it populates the Team menu with applicable CVS commands. To notify JBuilder,

1 Choose Team | Select Project VCS.

   The Select Project VCS dialog box appears:

2 Select CVS from the list of version control systems.

3 Click OK or press Enter to close the dialog box.

Placing a new project into CVS

Checking a project into CVS through JBuilder creates the directory structure in the repository, adds the files to the repository, and creates the CVS infrastructure needed to maintain revision management both in the repository and in your workspace.

Note If the project you are placing into CVS is part of a project group, and the project group file (.jpgr) is in the root directory of the project, the project group file will also be checked in to CVS. Additionally, when the project is
Placing a new project into CVS

pulled from the CVS repository, the project group will also be pulled from the repository and opened.

Checking in a project for the first time

From an open project,

1 Choose Team | Place Project Into CVS to bring up the Place Project Into CVS wizard.

The first step of this wizard configures the connection to the repository:

Under Connection Type, **Local** connects to a local repository, **PServer** connects to a repository on a server that’s normally password-protected, and **Ext** connects to a repository on a secure server. If you select PServer, you have the option of specifying a port number for connecting to the remote CVS repository host.

2 Enter your login settings according to the requirements of your connection type:

- A Local connection requires no login, since you’re already connected to the drive
- A PServer connection requires the name of the server you’ll connect to and your user name on that server
- An Ext secure server connection requires the server name, your user name, and the remote shell the server uses (usually ssh)

JBuilder fills in the user name you use on your machine by default. If your user name is different on the server, change this to match it. The special port number for PServer connections is required only if the CVS server has been configured to use a port number different from the default, 2401.
Placing a new project into CVS

3 Under Module Location, enter the path to the repository you want to place the new module in.

If you’re using a local repository, you may click the ellipsis (...) button to browse to it. Enter a new module name for the new module.

Note JBuilder maintains lists of prior connections for you to choose from. These lists are empty when you use the wizard for the first time, but after that you can select them from the drop-down lists instead of typing them in.

Tip The CVSROOT path appears at the bottom of the wizard. It reflects the information entered in this step. If you’re unsure what you should enter and you already have a CVS account on an existing server, check your machine’s user environment variables or see your CVS administrator.

4 Click Next to continue.

Step 2 provides space for a comment describing the new module.

5 Enter a comment and click Next to continue.

Step 3 allows you to exclude directories in the project from CVS:

6 Ensure the files you want to include in the repository are checked, and the files to exclude are unchecked.

Expand directories to fine-tune which files you want to include in each directory. Directories for backup and derived files are excluded by default. Source directories are included by default.

7 Click Finish to create a CVS module and check the project into it.

Note The project disappears from the AppBrowser while the project is checked in and then reappears once it’s under CVS control.
Creating a local repository

When you create a CVS module in JBuilder, the following things happen:

- JBuilder creates a module in the CVS repository to put a project into, and checks in the project.

  **Note**  
  The action of importing a project into CVS doesn’t change any other file in the existing directory structure.

- JBuilder checks out the newly created module (and project) to your workspace, allowing you to work with the files immediately.

  **Note**  
  JBuilder makes a backup copy of the original project directory by adding `.precvs` as a suffix to the directory name, and checks out the project to your workspace, using the name of the original directory. This precaution allows you to compare the contents of the checked out project directory with the contents of the original (.precvs) directory to ensure that all of the files you want to include in the CVS module are present and correct.

Creating a local repository

JBuilder provides a way to create a local repository. This is ideal for working on projects that don’t need to be shared but that would still benefit from revision management. A local repository gives you complete access and administrative control of the repository.

You can put as many modules as you want to into the repository, up to system allowances.

Use either an empty directory or enter a new one and JBuilder will create it for you. CVS’s administrative subdirectory is automatically created in the repository directory.

  **Warning**  
  Avoid using the directory as anything but a repository. Don’t put files directly into or take files directly out of the repository. It renders CVS useless in that repository, since CVS has no way of tracking changes that were not made through CVS.
Chapter 5

Programmer’s guide to the CVS integration

CVS is installed automatically inside the JBuilder root directory. No outside settings are made for it until you use the CVS integration. If you uninstall JBuilder, the version of CVS installed with JBuilder is uninstalled with it.

Once the integration is used, the environment variables for CVS are set and must be removed manually (if necessary.) Projects and repositories created after JBuilder is installed are protected: they will not be uninstalled with JBuilder.

This chapter provides in-depth material on the CVS integration and background information on CVS and configuration. It has two main sections:

• Guide to the CVS integration: discusses which commands can be accessed from the JBuilder user interface.

• Reference: describes how CVS handles binary files, environment variables for CVS, and how to use SSH for secure server connections.

Guide to the CVS integration

This guide discusses commands used in the CVS integration, so you can better understand what to expect from it. This part of the documentation is structured according to which commands are most closely associated.
There are four groupings used for these commands:

- Using CVS as your version control system: selecting CVS, configuring the repository connection, and creating a local repository
- Getting material out of the repository: pulling a project from CVS, checking files out of the repository, and updating
- Getting material into the repository: checking in files, checking in a project file, checking in an entire project, and committing
- Managing the repository: adding and removing files, version labeling, and using watches

The Status Browser and Commit Browser apply predictable commands, although they can apply them to many files quickly. The Status Browser passes `cvs status <filename1, filename2, ...>` to the command line, retrieving the status of each file from CVS itself.

The Commit Browser offers a menu of possible CVS commands for each altered file. It passes each command and its associated files to CVS, then commits each changed file and appropriately applies the individual and summary comments specified by the user.

In JBuilder SE and Enterprise, a file’s comments can be viewed Info page in the history pane.

**Using CVS as your version control system**

JBuilder executes CVS commands by passing them to the command line. This means that JBuilder can take advantage of all of CVS’s commands and options, just as, for the most part, JBuilder is bound by CVS’s limitations.

**Selecting CVS**

When you use Team|Select VCS and choose CVS from the dialog box, it tells JBuilder which set of Team menu commands to make available. It doesn’t pass any commands to CVS.

**Configuring the repository connection**

Repository connection configuration is built into the Pull Project From CVS and Place Project In CVS wizards. These wizards notify JBuilder of the connection configuration and, if necessary, write to your environment variables so CVS can find its administrative files.

If this is the first time CVS is configured on your machine, you can use CVS from the command line after you have configured a connection through JBuilder. However, if you do so, then JBuilder will have no way of being aware of CVS actions performed outside of its own environment.
Guide to the CVS integration

The CVS connection is set by environment variables. If the connection is configured properly in your environment variables before you use JBuilder’s Pull Project From CVS or Place Project Into CVS wizards, it is still necessary to use the connection configuration steps in the wizards so that JBuilder is fully aware of the repository connection used for each individual project.

When using a repository, it’s important to use it appropriately. Simply copying a project into your workspace prevents effective version control and keeps your changes out of the repository, where the project will continue to evolve regardless.

**Creating a local repository**

JBuilder uses `cvs init <repository_name>` at the level you specify in the path you set for the repository. It’s preferable to keep the repository at or near the user’s root.

This command is benign, so if you accidentally create the same repository twice in the same place, no files or data in that repository are overwritten.

**Getting material out of the repository**

When CVS is properly used, projects are checked in once per project and checked out once per user. After that initial checkin or checkout, the users maintain the congruency of workspace and repository versions by updating and committing changes. Once a project is checked in, it can’t be checked in again. Likewise, files can be added only once.

**Pulling a project**

When you run the Pull Project wizard, it checks a module out of the repository and opens it as a project in JBuilder. Access it either from the Project page of the object gallery (File|General) or, with a project open and CVS selected as the version control system, from Team|Pull Project From CVS.

For CVS, the wizard creates an administrative infrastructure and workspace:

1. If there are no environment variables pointing to the CVS `home` and `cvsroot` directories, it creates them.
2. It copies the project tree from the repository into the workspace.
3. It creates a subdirectory in each directory in the project and names it `cvs`. This is where CVS maintains its administrative files for the directory.

Items 2 and 3 are done using the `cvs checkout <modulename>` command.
For JBuilder, the wizard,
1. Records the project’s CVS parameters
2. Opens the module as an active JBuilder project
3. Makes the CVS menu items available from the Team menu and the project pane
4. Makes CVS version information available on the pages in the history pane

**Pulling a project file**
When you update a project file, JBuilder passes \texttt{cvs update <projectfilename>} to the command line. The project file is handled by CVS as a binary file to maintain the validity of the project file values. When you update or commit the project file, the old settings are overwritten. There’s no attempt to merge them with the new ones, since that could result in meaningless values.

As used in the CVS integration, the project file controls path settings: source, test, output, library, and backup paths. Local project settings, such as doc author and runtime configurations, are not affected by the synchronization. JBuilder handles this distinction behind the scenes.

Private settings, such as your running and debugging preferences, are kept in a local project file called \texttt{<projectname>.jpx.local}. This file is not affected by CVS. This means your local preferences can’t be overwritten by another user.

**Note** Only one project file is treated specially by the version control integration. By default, JBuilder uses the top-level project file in the current project directory. Child project files are checked in and out as normal binary files.

**Checking out files**
When you check out individual files, JBuilder passes \texttt{cvs checkout <filename>} to the command line at the working directory level. No options are used by JBuilder.

**Updating files**
When you update the active file from the Team menu or selected files from the project pane, JBuilder passes \texttt{cvs update <filename1>, <filename2>, ...} to the command line at the working directory level.

When you update files from the Commit Browser, JBuilder puts the \texttt{update} command and its list of files in the queue of commands passed from the Commit Browser to the command line.

Updating keeps the files in your workspace current, ensuring you work on the latest versions of the files. Updating also reduces the occurrence of
merge conflicts substantially. Updating before committing is the best way to avoid merge conflicts and the subsequent work involved in resolving them.

**Removing files**

JBuilder uses the `cvs remove <filename>` command to remove the file from CVS and then commits the removal automatically. It also removes the file from the JBuilder project.

If you don’t have a copy of that file somewhere else, this command will delete it entirely. You can retrieve it from storage in CVS, if necessary, by using the appropriate CVS command from the command line and then adding it to the repository again.

**Getting material into the repository**

Material is posted to the repository for the first time by checking in or adding. (Projects are checked in. Files are added.) Changes to existing material are posted to the repository by committing.

**Checking in a project**

The Place Project Into CVS wizard configures the connection to the repository for the project and creates a module for the project. If this is the first time CVS has been used on the machine, JBuilder writes the necessary environment variables.

JBuilder first creates a module based on the original project, then renames the original project by adding the `.precvs` suffix to the original project name, then it checks the new module back out into your workspace.

JBuilder creates the module using `cvs import -m "<comment text>" <module_name> <vendor_tag> <release_tag>`. Only the files you select in Step 3 of the wizard are checked in as part of the module. By default, JBuilder selects source material and the project notes file for checkin. JBuilder excludes generated files and backup files by default, since these files normally don’t need to be under version control. JBuilder also performs automatic package discovery so it can create package paths for you.

The project file is required by JBuilder to handle the project correctly. It must be in the project’s root directory. It must be checked in with the rest of the project initially, but is maintained separately from the rest of the project after that.

If there’s no project file, JBuilder creates one for you when you check the project in. If there is more than one project file, JBuilder asks which one you’d like to use as the top-level project file for version control purposes.

JBuilder adds the `.precvs` suffix to the original project name, so you have the opportunity to check the original project against the created module. This is common practice.
JBuilder checks out the new module back into your workspace so you can go to work on it immediately.

Adding new files
JBuilder adds a file to CVS using `cvs add <filename>`. If more than one file is added, the files are listed in a comma-separated string.

When you commit a file from the Team or project pane context menu, JBuilder does not commit the added files at the time, but allows you to commit them separately, once you’ve had a chance to use them and decide they should be kept. Use the Commit command, either from the menus or from the Commit Browser.

When you add a file from the Commit Browser, JBuilder passes `cvs add <filename>` to the command line. After adding the file, JBuilder automatically commits it behind the scenes, using `cvs commit <filename> -m [summary comment] [individual comment]`.

JBuilder automatically commits added files only from the Commit Browser. Files added from the menus must be committed in a separate step.

Committing changes
JBuilder passes `cvs commit <filename> -m [summary comment] [individual comment]` to the command line for each file committed, whether committed from the menus or from the Commit Browser.

Changes are committed behind the scenes for file removals and for files added from the Commit Browser. Files added from the menus and other file changes are committed only when the user explicitly uses the commit command from the IDE.

Posting a project file
JBuilder requires a project file of the .jpx type, for programmatic reasons that make it easier to work with in a shared development environment. The .jpx project file is an XML project file type.

To commit a project file, choose Team | Sync Project Settings. Choose Post Current from the submenu. JBuilder passes `cvs commit <projectfilename>.jpx` to the command line. The project file is treated by CVS as a binary file to maintain the validity of the project file values.

As used in the CVS integration, the project file controls path settings: source, test, output, library, and backup paths. Local project settings, such as doc author and runtime configurations, are not affected by the synchronization. JBuilder handles this behind the scenes.
When you update or commit the project file, the old shared settings are overwritten. There’s no attempt to merge them with the new ones, since that could result in meaningless values.

Managing the module

JBuilder supports two additional CVS features. Version labeling allows you to mark the evolution of an entire project. CVS Watches keep users informed about who else is using files.

Version labeling

JBuilder passes `cvs tag <tag_name>` to the command line from the project’s root directory. This tags all the files under CVS control that are in that directory and recursively tags all such files in its subdirectories.

Using watches

JBuilder’s CVS integration supports the use of watches, but it does not turn them on. The administrator (or someone with suitable access) must turn the watch on in the usual way to make these commands meaningful.

JBuilder uses the following console commands for the following menu items:

- Watches | Add Watch: `cvs watch add`
- Watches | CVS Edit: `cvs edit`
- Watches | CVS UnEdit: `cvs unedit`
- Watches | Remove Watch: `cvs watch remove`

To apply options to the watch and edit commands, it will be necessary to enter the commands at the command line.


CVS reference

This section provides information on binary files in CVS and how to set environment variables.

Handling binary files in CVS

If JBuilder does not yet recognize a non-text-based file type in CVS, you must specifically include the file’s extension in JBuilder’s list of Generic Binary file types. To include it,

1. Choose Tools | IDE Options.
2. Select the File Types tab.
3 Select the Generic Binary file type.
4 Click Add.
5 Enter the new extension.

If necessary, you can use the file stored in the .precvs directory to restore the original binary file.

Checking and setting user environment variables

The way you access environment variables depends on your platform: Linux, Solaris, Windows XP, Windows NT, or Windows 2000.

Your variables for CVS should include values for the following names:

- **cvs**: This is the directory that contains your CVS installation. It’s usually a top-level directory, so it’s probably at the root of your drive.
- **cvsroot**: This is the path to the server you use. See the following table for the syntax of this path.
- Once you have used CVS in JBuilder, JBuilder will create a variable called .cvspass. Please don’t change this variable.

Where needed, CVS wizards contain configuration pages. At the bottom of these pages, the CVSROOT variable appears, changing as you complete the configuration fields in the wizard. This is what the CVSROOT variable consists of in each of the types of server connections that JBuilder supports:

<table>
<thead>
<tr>
<th>Connection type</th>
<th>CVSROOT syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>:local:&lt;repository path&gt;</td>
</tr>
<tr>
<td>PServer</td>
<td>pserver:&lt;user name&gt;@&lt;server name&gt;:&lt;repository path&gt;</td>
</tr>
<tr>
<td>Ext</td>
<td>:ext:&lt;user name&gt;@&lt;server name&gt;:&lt;repository path&gt;</td>
</tr>
</tbody>
</table>

**Linux and Solaris**

Edit the PATH environment variable in your shell’s init file. Include,

- The path to the directory that contains your CVS installation
- The path to the repository on the server

Remember to separate each PATH variable with a semicolon.

**Windows XP**

Access your user environment variables from Start | Control Panel | System (using the Classic view) or Start | Control Panel | Performance And Maintenance | System (using the Category view.)
To change user environment variables,

1. Select the Advanced tab.

2. Click the Environment Variables button near the bottom of the dialog box.

3. Under System Variables, select the variable you want to change.

4. Click Edit to change an existing variable (such as PATH) or New to create a new one.

5. Confirm or enter a variable name and edit or enter a value for it.

6. Click OK or press Enter when done.

**Windows NT**

On Windows, your user environment variables are stored in the Settings directory. You can access this in two ways: from Start | Settings | Control Panel | System, or by right-clicking the My Computer icon on your desktop and choosing Properties.

In Windows NT,

1. Select the Environment tab.

2. Look under User Variables for the environment variables that pertain to CVS.

3. Select a user variable, and type a new variable name in the Variable field and enter a value in the Value field to create a new variable.

4. Click OK or press Enter when done.

**Windows 2000**

On Windows, your user environment variables are stored in the Settings directory. You can access this in two ways: from Start | Settings | Control Panel | System, or by right-clicking the My Computer icon on your desktop and selecting Properties.

In Windows 2000,

1. Select the Advanced tab.

2. Click Environment Variables in the middle of the dialog box.

3. Click Edit to change an existing environment variable or New to create a new one.
Using CVS with SSH

It is possible to use SSH (Secure Shell) to have a secure connection to a CVS server when using the Team Development features of JBuilder. SSH provides a way of connecting to a host machine using strong encryption, thus assuring that confidential data is kept confidential. JBuilder supports the use of SSH1 (SSH2 may work but has not been tested).

Caution

The connection must be configured so that SSH will not prompt for a password. Failing to configure SSH to connect without a password will cause JBuilder to “freeze” until the SSH program is explicitly terminated. Please refer to the SSH documentation for more information on configuring SSH connections.

To test if your connection is configured correctly, before trying it in JBuilder, enter `ssh` followed by the name of the server you want to connect to. For example, if the server is called `codecentral`, enter:

```
ssh codecentral
```

If SSH connects to the server without prompting you for a password, you can use SSH in JBuilder.

Configuring the SSH connection in JBuilder

After SSH is properly configured, it can be used within JBuilder for placing projects into the CVS repository, and any subsequent CVS operations. Connection parameters are set for projects when they are initially placed into CVS, and cannot be adjusted thereafter.

To configure CVS to use SSH when placing a project into CVS,

1. Choose Team | Place Project Into CVS.
2. In the Place Project Into CVS wizard, click the Ext radio button in the Connect Type section.
3. Enter the name of the server in the Server field, and enter your user name on that server in the Username field.
4. Type `ssh` in the Remote Shell field.
5. Complete the rest of the steps in the wizard to place your project into CVS.

Once a project has been placed into CVS using SSH, all subsequent CVS operations for the project will use SSH.
JBuilder’s integration of Visual SourceSafe (VSS) allows you to perform the most common version control tasks from within the development environment.

JBuilder’s integration of VSS is designed and tested on Visual SourceSafe version 6.0. This integration is supported on Windows NT and 2000.

You must have the Visual SourceSafe client installed on your machine to access the Visual SourceSafe menu commands from within JBuilder. To connect to an existing VSS database, you must have LAN access to the directory that the VSS database is located on. If you can see the VSS directory in Windows Explorer, you should be able to access it in JBuilder.

Visual SourceSafe commands are generally available from two places: the Team menu from the main menu bar, and the context (right-click) menu in the project pane.

VSS commands other than those described in this documentation must be executed from within VSS Explorer or at the command line. Invoke VSS Explorer from the active project by choosing Team | Visual SourceSafe Explorer.

Configuring the connection

Place Project Into VSS (available from the Team menu) and Pull Project From VSS (available from the Team menu or the object gallery) are wizards that configure your connection for the project you want to work on. Each project’s configuration is set individually.

When you install JBuilder, it looks for an installation of Visual SourceSafe. If it finds one, it configures its path to the local VSS installation and to the
default database. If not, JBuilder prompts you to enter this information when you first use either of these wizards.

The wizards guide you through the process of setting the connection to the database, setting up your working folder, and choosing which files to keep checked out. Once your connection is configured for that project, you don’t have to set it again.

Note Performance improves greatly when the VSS client is installed on a local file system rather than on a local area network (LAN).

View and test the configuration by choosing Team | Configure Visual SourceSafe. The Configure Visual SourceSafe dialog box appears:

![Configure Visual SourceSafe dialog box]

Note The username and password fields are writable. If your user name or password for Visual SourceSafe changes, choose Team | Configure Visual SourceSafe and change these fields to match your current identification data. Click the Test button to check the configuration before you actually use it.

See also

- “Configuring the database connection” on page 9-2
VSS integration is a feature of JBuilder Enterprise

JBuilder supports common VSS tasks and provides features that make applying commands to multiple files much easier.

## Pulling an existing project

There are two possible ways to pull a project from the database from within JBuilder: from the object gallery, or from the Team menu.

**From the object gallery,**

1. Choose File | New to open the object gallery.
2. Select the Project tab.
3. Select Pull Project From VSS.
4. Double-click the icon, click OK, or press Enter to start the Pull Project From Visual SourceSafe wizard.

**Note** This is the only VSS command available from the object gallery.

**From the Team menu,**

1. Open or create a project.
   - The Team menu is not activated unless a project is open in the AppBrowser. Use any project, as it will not be affected by version control commands that don’t apply to it.
2. Choose Team | Pull Project From VSS.
Pulling an existing project

Both of these methods bring up the Pull Project From Visual SourceSafe wizard. This wizard configures your connection and sets up everything you need to get to work on a project that’s under Visual SourceSafe control.

When you install JBuilder, it searches for a VSS installation. If it finds one, it records the location of the local installation and chooses a default database. If JBuilder doesn’t find these paths, this wizard prompts you for them. In that case, this wizard has five steps. The first step tells JBuilder where to find the Visual SourceSafe client runtime directory:

![Select Visual SourceSafe runtime directory](image)

**Tip** If you happen to choose an invalid directory, a message appears in the wizard letting you know that the selection is not valid. In that case, click the ellipsis (...) button and try again.

For every project you put under Visual SourceSafe after that, the Pull Project from Visual SourceSafe wizard will have four steps:

2. Enter Username And Password
3. Select Visual SourceSafe Project
4. Select An Empty Target Directory

**Note** JBuilder does not change your access rights in Visual SourceSafe. To execute a VSS command from within JBuilder, you must first have suitable user rights in VSS.

---

**Selecting the database directory**

The Select Visual SourceSafe Database Directory step is Step 1 once the path to the runtime directory is set.

The first time you access Visual SourceSafe through JBuilder’s integration, this field is empty. Click the ellipsis (...) button to browse to the database directory. The database directory is where the projects are stored.
Once you have accessed a database directory, that becomes your default database directory. It appears in the box, but you can still select others either by browsing to them using the ellipsis (...) button or by clicking the down arrow and selecting from the list of database directories you have previously used.

**Entering security information**

The Enter Username And Password step ensures that you have the necessary access to VSS. This information is not persisted, so you must enter it each time you use the wizard.

**Selecting a project**

The Select Visual SourceSafe Project step provides a drop-down list of the projects that are available in that database directory in a drop-down list.

**Choosing a target directory**

The Select An Empty Target Directory step asks for an empty directory to pull the project into. The project is pulled in as a subdirectory of that directory.

**See also**

- The Help button in any page of the wizard.
- “Configuring the connection” on page 9-1 to learn more about how to maximize performance and how the wizard handles default paths.

### Checking out files

The Check Out command copies the selected file or files from the current VSS project to the folder in your work area and makes them writable to you.

To check out the active file,

1. Choose Team | Check Out “<filename>”.
   
   A Check Out File dialog box appears, listing the active file.

2. Click OK to check out the file.

3. Click OK to close the dialog box.

**Note**

By default, JBuilder displays dialog boxes to confirm the success or failure of version control system (VCS) operations. To configure these
Confirmation dialog boxes to close automatically after successful VCS operations, check Close VCS Dialogs Automatically After Successful Operation in the IDE Options dialog box (Tools | IDE Options).

To check out files by using the project pane context menu,
1. Select the file or files in the project pane.
   Use the Ctrl key or the Shift key to select more than one file.
2. Right-click the selection, and choose Visual SourceSafe from the context menu.
   The menu that appears applies to all of the selected files.
3. Choose Check Out Files from the menu.
   A Check Out Files dialog box appears. Look at the list of files and make sure it matches what you intended.
4. Click OK to check out the files.
5. Click OK to close the dialog box.
You must have the Check Out access right to use this command.

**Undoing a checkout**

Undoing a checkout cancels the checkout of selected files, voiding all changes.

To undo the checkout of the active file,
1. Choose Team | Undo Check Out “<filename>”.
   An Undo Checkout dialog box appears, listing the active file.
2. Click OK to undo the checkout on the file.
3. Click OK to close the dialog box.

To undo the checkout of files by using the project pane context menu,
1. Select the file or files in the project pane.
   Use the Ctrl key or the Shift key to select more than one file.
2. Right-click the selection, and choose Visual SourceSafe | Undo Checkout from the context menu.
   An Undo Checkout dialog box appears. Look at the list of files and make sure it matches what you intended.
3. Click OK to undo the checkout on the files.
4. Click OK to close the dialog box.
Whether the latest database version of the file is pulled into your workspace depends on a Visual SourceSafe variable that your administrator sets. Undoing a checkout may simply revert any changes you made, so that the version of the file you see is the same one you checked out.

You must have the Check Out access right to use this command.

See also

* “Undoing a checkout” on page 9-4

## Checking out a project file

Pulling the latest .jpx project file checks out the most current shared .jpx project file for the active project from the VSS database.

**Note**

Only the active project file in the current project directory is treated specially by the version control integration. All other project files are treated by version control as binary files, so that they are overwritten rather than merged.

To pull the latest project file from the Team menu,

1. Choose Team|Pull Latest “<project file name>.jpx”.
   
   The Visual SourceSafe Pull Project File dialog box appears.
2. Click OK to pull the project file.
3. Click OK to close the dialog box.

To pull the latest project file by using the project pane context menu,

1. Right-click the project file node, and choose Visual SourceSafe|Pull Latest “<project file name>.jpx” from the context menu.
   
   The Visual SourceSafe Pull Project File dialog box appears.
2. Click OK to pull the project file.
3. Click OK to close the dialog box.

The project file maintains project settings, such as paths and library configurations. This information is shared by other users of the project. JBuilder allows you to pull and post the project file separately to help keep it current and available to other users as much as possible.

JBuilder protects local project settings, such as doc author and runtime configurations, so you can pull and post the project file as much as necessary without overwriting local project settings.
Checking in files

See also

- “Creating and managing projects” in *Building Applications with JBuilder* to understand project files better.
- “Checking the project file in or out” on page 9-4

Checking in files

Checking in files updates the database with changes you made to the checked-out file or files. This dialog box offers the option of keeping the file checked out so you can continue to work on it after checking in your changes. If you decide not to keep it checked out and you work under a locking VSS system, this command unlocks the VSS master copy of that file so that someone else can write to it.

**Note**

With multiple users are working on the same files, VSS performs an automatic merge when changes to the same file are checked in by more than one user. Merge conflicts can occur when files are checked in if the same line of code has been modified by two or more different users. See “Reconciling merge conflicts” on page 7-8 for more information, including instructions for resolving conflicts.

To check in the active file,

1. Choose Team | Check In “<File name>”.
   A Check In Files dialog box appears, listing the active file.
2. Type a comment describing the changes, according to your usual practices.
3. Click OK to check in the file.
4. Click OK to close the dialog box.

To check in files by using the project pane context menu,

1. Select the file or files in the project pane.
   Use the *Ctrl* key or the *Shift* key to select more than one file.
2. Right-click the selection, and choose Visual SourceSafe | Check In Files from the context menu.
   A Check In Files dialog box appears. Look at the list of files and make sure it matches what you intended.
3. Type a comment describing the changes, according to your usual practices.
4 Click OK to check in the files.
5 Click OK to close the dialog box.
You must have the Check Out access right to use this command.

**Checking in the project file**

Posting the current .jpx project file checks in the most current shared .jpx project file for the active project into the VSS database.

**Note**
Only the active project file in the current project directory is treated specially by the version control integration. Other project files in the project are treated by version control as binary files, so they are overwritten rather than merged.

To post the latest project file from the Team menu,
1 Choose Team | Post Current “<project file name>”.
   The Visual SourceSafe Post Project File dialog box appears.
2 Type a comment describing the changes, according to your usual practices.
3 Click OK to check in the project file.
4 Click OK to close the dialog box.

To post the current project file using the project pane context menu,
1 Right-click the project file node.
2 Choose Visual SourceSafe | Post Current “<projectfilename>” from the context menu.
   The Visual SourceSafe Post Project File dialog box appears.
3 Type a comment describing the changes, according to your usual practices.
4 Click OK to check in the project file.
5 Click OK to close the dialog box.

The project file maintains key project paths and parameters. This information is shared by other users of the project. JBuilder allows you to pull and post the project file separately to keep it available to other users as much as possible.

See “Checking the project file in or out” on page 9-4 to learn more about checking in project files, and “Creating and managing projects” in *Building Applications with JBuilder* to understand projects and project files.
Reconciling merge conflicts

JBuilder helps avoid merge conflicts by requiring you to update when necessary before committing changes. If merge conflicts do occur, VSS and JBuilder alert you in the following ways:

- JBuilder displays a conflict message in the message pane. Click a merge conflict warning within the message pane to display the Merge Conflicts page in the history pane.
- VSS inserts “<<<<<<<”, “======”, and “>>>>>>” around the conflicting blocks in the file in your workspace. For example, if you modified a comment in the file in your workspace, but someone else modified the same comment and checked the changes in, the conflict might be tagged like the following code sample:

```
<<<<<<< Frame1.java
// Comment modified in file in the workspace
======
// Comment in file checked into repository
>>>>>> 1.3
```

This tagging is caught by the compiler, so if you postpone resolving conflicts in Java files, you can find them again later by compiling the file and using the message pane.

- JBuilder’s Merge Conflicts page in the history pane displays the workspace source side-by-side with the repository source, with the conflicting blocks of code or text highlighted. Radio buttons next to the highlighted blocks of conflicting code allow you to easily select which block of code you want to keep. The preview pane at the bottom of the Merge Conflicts page shows what your workspace file will look like when you apply the changes.

There are two ways to reconcile VSS merge conflicts in JBuilder:

1. Reconcile them automatically, using the radio buttons in the Merge Conflicts page in the history pane to choose the code block to keep. The Merge Conflicts page is enabled only when JBuilder detects a conflict.

2. Reconcile them manually, searching for the “<<<<<<<”, “======”, and “>>>>>>” tagging in the file source, and using the editor to resolve the conflicts. Remember to delete the conflict tagging and any extraneous text when you’re done.

Choose the way that best addresses the conflict.
Reconciling conflicts automatically

JBuilder’s automatic merge conflict handling mechanisms are suitable for relatively straightforward conflicts where sequential changes were made to the same block of text.

To choose automatically between old and new versions of a conflicting area of text,

1. Go the history pane for the file, and open the Merge Conflicts page.

   The conflicting areas in the workspace source and repository source are highlighted in separate panes on the Merge Conflicts page. Radio buttons in the gutters of the workspace and repository panes are used to select the changes to keep.

   Lines for a conflict that has been selected to be kept are highlighted in yellow, and are further indicated by plus (+) signs in the gutter. Changes to be discarded are highlighted in red and have minus (-) signs in the gutter. The preview pane at the bottom of the Merge conflicts page shows what the file source will look like when the changes are applied. By default, the changes in the workspace are selected to be retained.

   If there are multiple conflicts, you can click the navigation arrows at the bottom left corner of the page to scroll forward and back between pairs of conflicts.

2. Click the radio button next to the version (workspace or repository) of the block of text that you want to keep.

   When you click a radio button, the highlight color changes to yellow, and a plus (+) sign appears in the gutter of the selected line or lines. The preview pane updates to show how the file will look if the changes are
Reconciling merge conflicts

applied. The icon in the gutter of the preview pane changes to indicate the source (workspace or repository) of the selected change.

3. Go to the Source pane to apply the change in the editor buffer.

Applying changes updates the editor buffer for the file in the workspace. Until you commit your changes, you can use the undo command (Edit | Undo or Ctrl+z) to restore the conflicts. The undo command is not available from the history pane, so you will need to switch to a different view.

4. Save the file and commit the changes.

Reconciling conflicts manually

If you don’t want to use JBuilder’s automatic merge conflict handling mechanisms, you can edit files manually to resolve conflicts.
To manually reconcile conflicts,

1. Search for the "<<<<<<", "======", and ">>>>>>" tagging in the file source.

```
package thisproject;
// second change
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
<<<<<< Frame1.java
// fifth change
======
// conflicts with the fifth change
>>>>>> 1.5
```/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author H. Marx
 * @version 1.0
 */

2. Edit the text as you normally would until the conflict is resolved.

```
package thisproject;
// second change
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
// conflicts with the fifth change
======
// conflicts with the fifth change
```/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author H. Marx
 * @version 1.0
 */

Be sure to remove the conflict tagging and any extraneous text.

3. Save the file and commit the changes.

**Note** Once conflicts have been tagged, VSS no longer sees them as conflicts.

**Tip** The compiler will report errors if it finds conflict tags when it compiles. Therefore, if you postpone resolving a set of conflicts, you can compile and then double-click the compiler’s error message to find the conflicts in the editor.
Adding and removing files

JBuilder supports the Add command and the Remove command.

Adding files

Adds files into the VSS database and logs the comment you type for the addition.

To add the active file,

1. Choose Team | Add "<filename>".
   An Add Files dialog box appears, listing the active file.

2. Type a comment describing the additions, according to your usual practices.

3. Click OK to add the file.

4. Click OK to close the dialog box.

To add files by using the project pane context menu,

1. Select the file or files in the project pane.
   Use the Ctrl key or the Shift key to select more than one file.

2. Right-click the selection, and choose Visual SourceSafe | Add Files from the context menu.
   An Add Files dialog box appears. Look at the list of files and make sure it matches what you intended.

3. Type a comment describing the additions, according to your usual practices.

4. Click OK to add the files.

5. Click OK to close the dialog box.

VSS uses its AutoDetect feature to specify whether the added files are text or binary files.

You must have the Add access right to use this command.
Removing files

Removes files from VSS Explorer and marks them as deleted. The items still exist, however, and can be recovered using the Recover command (accessible from Visual SourceSafe).

To remove the active file,
1. Choose Team | Remove “<filename>”.
   - A Remove From Visual SourceSafe dialog box appears.
2. Click OK to remove the file.
3. Click OK to close the dialog box.

To remove files by using the project pane context menu,
1. Select the file or files in the project pane.
   - Use the Ctrl key or the Shift key to select more than one file.
2. Right-click the selection, and choose Visual SourceSafe | Remove Files from the context menu.
   - A Remove From Visual SourceSafe dialog box appears. Look at the list of files and make sure it matches what you intended.
3. Click OK to remove the files.
4. Click OK to close the dialog box.

You must have the Delete access right to use this command.

Checking in the entire project

JBuilder provides tools that display the status of all the files you have altered in a project and that let you apply individual version control commands to all of the altered files in a project at once.

Browsing the project’s status

The Status Browser (Team | Status Browser) displays all the files in your project and the version control status of each one. It has two pages: the Changes page and the File Include Lists page.

In the Changes page, the directory tree view is in the left-hand pane and the list of files is in the right. Select the Full List node to view a list of all the files in the project. Select a directory node to see a list of the files in that directory.
Checking in the entire project

The version control status of each file is noted in the Status column of the list.

Select a file in the file list. The tabbed pane below the file list allows you to view the source code of the selected file in any way that’s pertinent. For instance, if you choose a file that has been changed locally, the Workspace Source, Repository Source, and Workspace Diff tabs become available. If you choose a file that has been changed on the database, the Workspace Source, Repository Source, Repository Diff, and Complete Diff tabs become available.

Table 7.1 Changes page source views

<table>
<thead>
<tr>
<th>Source view</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workspace Source</td>
<td>This file’s source code from the current workspace version.</td>
</tr>
<tr>
<td>Repository Source</td>
<td>This file’s source code from the current database version.</td>
</tr>
<tr>
<td>Workspace Diff</td>
<td>This file’s most recent changes in your workspace.</td>
</tr>
<tr>
<td>Repository Diff</td>
<td>This file’s most recent changes in the database.</td>
</tr>
<tr>
<td>Complete Diff</td>
<td>Differences between the current version of this file in the database and the current version in your workspace.</td>
</tr>
</tbody>
</table>

Choosing files to include

The Status Browser displays and determines which files remain under version control and which files remain strictly local. It also filters files that remain under version control but are not necessarily of interest to you. It has two pages: Team Include List and Personal Include List.
The Team Include List shows which files are under version control and which are local. These files are tracked by the `<projectname>.jpx` file. The information you enter here is stored in that file as well, because this is a team-wide project setting. The files that are checked in the Team Include List are the files that everyone needs to be able to use.

The `bak` files and the `<projectname>.jpx.local` file are normally excluded (unchecked). Check your company’s policy about what files should be included and what should be excluded from a team project checkin.

**Caution**

The shared `.jpx` file must be included (checked in the Team Include List) to maintain version control for the project with VSS in JBuilder.

The Personal Include List filters your personal view, so that only the files you specify can show up in the Changes page. These files are tracked by the `<projectname>.jpx.local` file. The Personal Include List is entirely for your convenience. Since you won’t necessarily be working on every file in the project, you don’t necessarily want to look at all of them whenever you do a checkin.

This simple chart illustrates the concept of available files as compared to the files needed by an individual developer:

```
EntireProject
  - UIFiles
    - UIFile1
    - UIFile2
  - DataFiles
    - DataFile1
    - DataFile2
    - DataFile3
    - DataFile4
  - Beans
    - Bean1
    - Bean2
    - Bean3

Only the files
this engineer
wants to see
```

The Personal Include List page lets you keep in view only the files that you need. The rest stay hidden until you want to see them again.

**Checking in the project**

To check in changes for every modified file in the project at once, use the Commit Browser (Team | Commit Browser). The Commit Browser provides status messages and context-sensitive default VSS actions for each modified file in the project.

The Commit Browser also provides alternative VSS commands in drop-down menus in the Action column. The drop-down menus are context-sensitive; available commands vary depending on the status of the file. The default option varies depending on how the file achieved that status.
The Commit Browser has three pages: Commits, Summary Comment, and File Include Lists. It opens to the Commits page by default.

An expandable directory tree view is in the left-hand pane and the list of files is in the right. Expand the tree view to display subdirectories and directory contents.

In the Commits page, tabbed panes display the selected file's source in five ways:

<table>
<thead>
<tr>
<th>Tab</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workspace Source</td>
<td>This file's source code from the current workspace version.</td>
</tr>
<tr>
<td>Repository Source</td>
<td>This file's source code from the latest database version.</td>
</tr>
<tr>
<td>Workspace Diff</td>
<td>This file's most recent changes in your workspace.</td>
</tr>
<tr>
<td>Repository Diff</td>
<td>This file's most recent changes in the database.</td>
</tr>
<tr>
<td>Complete Diff</td>
<td>The differences between the current local version of this file and the latest database version.</td>
</tr>
</tbody>
</table>

**Action options**

JBuilder chooses default options to place in the Action column based on the file's status as reflected in the Status column. For changes made within JBuilder, the default actions are chosen not only according to the file's status but how it reached that status. For instance, if a file isn't in the workspace, it might be because it was removed from the project or because it has not yet been checked out. JBuilder senses the reason that it's
not in the workspace and chooses the most likely option to list as the default: Remove From Repository or Check Out. Additions and removals made from this browser are automatically checked in.

The list of files has a context menu that supports multiple selections. To apply the same option to several files, select the files you want, using the Shift key or the Control key to extend the selection beyond the first file selected. Right-click any of the selected files, then choose the option you want from the context menu.

If you want to check in some but not all changed files in the project, select No Action as the option for the files you don’t want to check in. You can also auto-select No Action for all of the files in a directory by right-clicking the directory in the node tree view and choosing Perform No Action For All In Directory from the context menu.

It’s best to add and delete files from within JBuilder, so JBuilder can perceive the reason for the changed status of the files and select the most appropriate default option. The following table lists common options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>Check in the change to the version control database.</td>
</tr>
<tr>
<td>Add</td>
<td>Add this file so it’s stored in the database.</td>
</tr>
<tr>
<td>Delete</td>
<td>This file has already been removed from the database; this removes it from your workspace.</td>
</tr>
<tr>
<td>Get</td>
<td>This file has already been added to the database; this checks it out to your workspace.</td>
</tr>
<tr>
<td>Undo Checkout</td>
<td>Update the workspace with the latest database version of this file, discarding all changes made since your last update.</td>
</tr>
<tr>
<td>No Action</td>
<td>When you click OK, this file will not be touched by a version control operation of any kind. It will be exactly as you left it before you invoked the Commit Browser.</td>
</tr>
</tbody>
</table>

The following table demonstrates a few status notations and associated options.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Listed status</th>
<th>Default option</th>
<th>Alternative options</th>
</tr>
</thead>
<tbody>
<tr>
<td>File changed in the workspace</td>
<td>Changed In Workspace</td>
<td>Commit to Repository</td>
<td>• Undo Checkout</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No Action</td>
</tr>
<tr>
<td>File added to version control</td>
<td>Not In Repository</td>
<td>Add to Repository</td>
<td>• Delete Locally</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No Action</td>
</tr>
<tr>
<td>File deleted from version control</td>
<td>Not In Workspace</td>
<td>If removed from within JBuilder: Remove From Repository</td>
<td>• Undo Checkout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If removed from outside JBuilder: Get From Repository</td>
<td>• No Action</td>
</tr>
</tbody>
</table>
Version labeling a project

Note The term repository is used as a generic term for the version control structure that VSS calls the database. In JBuilder’s version control terminology, database and repository are synonymous.

Summary comments
The Use Summary Comment checkbox at the bottom of the Individual Comment pane allows you to attach the same summary comment to multiple files. This summary comment will be maintained along with the individual comment written for each file. Select the Summary Comment tab to write a summary comment.

The Summary Comment page offers the option of placing the summary comment before the individual comments applied to individual files. This option is on by default. If you deselect this option and select the Use Summary Comment option on the Commits page, then the summary comment will be applied only to files that don’t have individual comments.

Version labeling a project

Version labels mark the evolution of the entire project without reference to the changes in individual files. To create a version label for the active project,

1 Choose Team | Create Version Label.
   The Create Visual SourceSafe Version Label dialog box appears.

2 Name the version label according to your usual practices and enter a comment describing the label’s purpose.

3 Click OK to close the dialog box.
   JBuilder applies the label to every file in the project that resides on the database.

See also
• “Version labeling” on page 9-6 for usage guidelines.
Working on a new project in VSS

VSS integration is a feature of JBuilder Enterprise

JBuilder must be notified that you want to put the active project under version control. Then it populates the Team menu with applicable VSS commands. To notify JBuilder,

1. Choose Team | Select Project VCS.
   
   The Select Project VCS dialog box appears:

   ![Select Project VCS dialog box]

2. Select Visual SourceSafe from the list of version control systems in the dialog box.

3. Click OK or press Enter to close the dialog box.

Placing a new project into Visual SourceSafe

Once you have selected Visual SourceSafe as the version control system for a project, you can place the project into a Visual SourceSafe database with the Place Project Into VSS command on the Team menu. Placing a project into a VSS database enables version control of that project and makes it accessible to other users of the version control system.

The Place Project Into VSS command invokes the Place Project Into VSS wizard. With this wizard, you take an open JBuilder project and create a
Placing a new project into Visual SourceSafe

corresponding VSS project in an existing VSS database. In the wizard, you specify which files and subdirectories to include in the VSS project and which files to keep checked out after the project is placed into VSS.

When you install JBuilder, it searches for a VSS installation. If it finds one, it records the location of the runtime directory and chooses a default database. If JBuilder doesn’t find these paths, this wizard prompts you for them.

To place a project into VSS,

1. Choose Team | Place Project Into VSS.

   The Select Visual SourceSafe Database Directory page appears. After the first use, the first database you used is offered as the default. Choose a different one from the drop down menu or browse to a new location.

   If you don’t know your VSS database directory path and it doesn’t appear to match the hints provided, please see your VSS administrator.

2. Click Next to go to the Enter Username And Password page.

3. Enter the identification data you use to access Visual SourceSafe.

   If your user name or password changes in Visual SourceSafe, you can change it to match by choosing Team | Configure Visual SourceSafe.

4. Click Next to go to the Select Directories And Files To Include page:

   ![Select Directories And Files To Include](image)

   Check the directories that you want to include entirely. If you want to choose individual files to include, expand the directory in the tree view and check the files inside it that you want to keep.

   By default, the bak and classes directories are excluded and the src files are included. The .jpx project file is required to be included because it
Placing a new project into Visual SourceSafe

manages team-wide version control settings and preferences as well as other project settings.

5 Click Next to go to the Select Files To Keep Checked Out page:

Expand the directories to see the files inside. Check the files you want to keep checked out to your work area so you can work on them after the project is placed into VSS.

6 Click Next to proceed to the Select Location In VSS Database page.

7 In the Location field, select an existing root VSS directory to keep the new project in.

All of the root directories available in the database are listed in the drop-down menu.

8 In the Project Name field, type a unique project name for this project. This will be the project directory inside the database’s root directory.

9 Click Finish.

The Placing Project Into VSS feedback dialog box appears. It reports on the progress of the command and lets you know when the project is in VSS.

10 Click Close to close the dialog box and return to the IDE.

The files you checked out appear in the content pane with write access assigned to you until you check them back in.

See also

• “Pulling an existing project” on page 7-1 to learn more about runtime directory discovery.
VSS integration is a feature of JBuilder Enterprise

The JBuilder integration of VSS is based on VSS’s command-line interface. This provides compatibility with different versions of VSS. This chapter includes background information on the VSS integration including some commands, technical information, and suggestions to improve performance.

Configuring the connection

A project’s connection to the version control system is configured the first time you access that project in the repository through JBuilder’s interface. This happens when you pull a project from the repository or post a project to the repository for the first time in JBuilder. After the connection has been set, you can view and modify it from the Configure Version Control dialog box.

Selecting VSS

Using Team | Select Project VCS and selecting Visual SourceSafe notifies JBuilder which set of version control menu commands to make available. This command notifies VSS and populates the Team Development menus with appropriate commands.

See also

- “Runtime location and performance” on page 9-2 to understand how to improve the performance of this command.
Configuring the database connection

When you first install JBuilder, it looks for a Visual SourceSafe installation. If it finds one, it sets the path to the VSS installation and the default database. If you install VSS after JBuilder (or if JBuilder can't find the VSS installation for some reason) then, the first time you use the Pull Project or Post Project wizard, there's an extra step that prompts you for those paths. Once the paths are set, JBuilder stores them for all future projects.

If these paths need to be set,

- Step 1 of the wizards, Select Visual SourceSafe Runtime Directory, prompts you for the location of the client installation, ss.exe.
- Step 2 of the wizards, Select Visual SourceSafe Database Directory, prompts you for the parent directory of the database.

Once the paths are set, the Select Visual SourceSafe Database Directory page becomes Step 1. It offers the path to the default database directory, and allows you to choose or type in another one.

Runtime location and performance

The Select Visual SourceSafe Runtime Directory step notifies you that a local installation of the VSS client improves performance significantly. This is due to limitations in technology, particularly the inevitable time-lag required to send and receive a large number of VSS commands simultaneously through a LAN connection. This time-lag is most noticeable when you first access VSS through JBuilder at the start of a work session. Sending commands to and receiving responses from the local machine is far faster.

You can install VSS locally through the LAN. Visual SourceSafe states that this is the preferred way to do so, as it configures your VSS connection properly and saves you time. Consult your VSS administrator to learn how to install the client locally from the LAN.

Pulling an existing project

Pulling an existing project writes the latest database version of the project into your work area. Once the project has been pulled, use the menus to check out the files you want to work on.

Selecting the Visual SourceSafe database directory

The Select Visual SourceSafe Database Directory step in the Pull Project From Visual SourceSafe wizard tells JBuilder which database directory to start in. The database directory is the parent directory that contains the
Pulling an existing project

databases stored in Visual SourceSafe. The term “repository” used in the Commit Browser and Status Browser means the same as database directory when using the Visual SourceSafe integration.

After the initial use,

- The Select Visual SourceSafe Database Directory step is Step 1.
- The database directory you selected the first time becomes the default in the Directory field.

Any time you use this wizard, the drop-down list and the ellipsis (…) button are available, allowing you to choose or navigate to a different database directory.

### Entering user name and password

The Enter Username And Password step of the Pull Project From Visual SourceSafe wizard provides JBuilder with your identification information, which it passes to VSS to enable your access through the JBuilder interface.

### Selecting a Visual SourceSafe project

The first time you use the Pull Project From Visual SourceSafe wizard, the Project field of the Select Visual SourceSafe Project step will be empty. Click the down arrow to choose from the list of projects in that database. JBuilder may take a moment to query the database and return the full list of the projects available.

The first project chosen becomes the default subsequently displayed in this field. All the other projects in that database are always available from the drop-down list.

### Selecting an empty target directory

You need a work area to pull the project into. The Select Empty Target Directory step defines that work area as a directory which you name. The project is pulled in as a subdirectory of the target directory. In navigating to the project in the future, keep this extra layer in mind.

The final page of the wizard displays stdout output from the commands passed to VSS.

### See also

- “Configuring the database connection” on page 9-2 to learn more about the first steps of this wizard.
Checking out files

Checking out a file writes the latest version of the file to your work area. VSS can be set to function either as an optimistic system, allowing multiple checkouts and merging differences, or as a pessimistic system, allowing only one user at a time to write to a file. It depends on company practices and certain settings the VSS administrator makes.

JBuilder respects the read/write status that the administrator sets for the checkout command. Therefore, if checking out a file normally means that others can’t write to it, then checking out a file using JBuilder still means others can’t write to it. Likewise, if several developers can write to the same file simultaneously, then they can still do so when they use JBuilder to check the file out.

Undoing a checkout

Undoing a checkout voids all changes you made to that file since you checked it out. Database settings determine whether the latest version is retrieved from the database or the local version is simply reverted.

Undo Checkout performs a Get operation when it’s finished, unless the Delete_Local initialization variable in the ss.ini file is set to Yes.

If someone checks in a file that has not changed and inadvertently creates a new version, VSS performs an Undo Check Out automatically. This way the file is unlocked and therefore accessible again, but no new version is logged.

Checking in files

The checkin command posts your changes to the database version of the files, generating a new version of each file. If the files were locked while they were checked out to you, the lock is removed. This depends on an administrative setting in VSS. If you have questions about it, please ask your VSS administrator.

Important

The current top-level .jpx project file for the project is checked in and out by a separate process. Please see the next heading.

Checking the project file in or out

The project file is checked in and checked out separately from the rest of the project. It’s handled slightly differently from other files under VSS control.
This command checks it in without a lock, regardless of the administrative settings regarding regular checkouts. Therefore, the project file will always be able to be checked out and changed by multiple users. Keeping it synchronized is important.

JBuilder’s version control integrations require a project file of the .jpx type, for programmatic reasons that make it easier to work with in a shared environment. The .jpx project file is an XML file type. However, to maintain the validity of the project file values, the .jpx file is handled by VSS as a binary file. When you check in or check out the project file, the old settings are overwritten. There’s no attempt to merge them with the new ones, since that could result in meaningless values.

As used in the VSS integration, the project file controls path settings: source, test, output, library, and backup paths. Local project settings, such as doc author and runtime configurations, are not affected by the synchronization. JBuilder handles this distinction behind the scenes.

Note
Only the active project file in the current project directory is treated specially by the version control integration. Child project files are checked in and out as normal binary files.

Adding and removing files

As you work on a project, you will likely add and remove files. In turn, these files must be added and removed from the Visual SourceSafe database to maintain version control for the project.

Adding files

Adds files to the VSS database and logs the comment you type for the addition. JBuilder uses a simple \texttt{ss -add} command for each file, so if you want to apply options, use the command line.

Removing files

Removes files from the VSS database. Logs the comment you type. JBuilder uses an unadorned \texttt{ss -remove} command for each file, so if you want to apply options, use the command line.

Keep two things in mind:

- Files removed this way can be recovered using VSS’s Recover command. This command is not supported by JBuilder out of the box.

- Files are removed only from the database, not from the JBuilder project. To remove files from the directory and no longer see their nodes in the project pane, use the Remove From Project command in the project pane.
Checking in the entire project

JBuilder provides browsers that can display pertinent version control information and execute commands on many files at once. Both the Status Browser and the Commit Browser query VSS for the status and repository version of each changed file, and combine that information with information that JBuilder already has from your work session.

All the VSS commands that are used by these browsers have already been discussed. They do exactly what you’d expect, based on the information presented so far: Add passes an unadorned -add command, and so on. The purpose of the browsers is to maximize productive time and minimize time spent dealing with version control responsibilities by bringing together all of the supported VSS commands and the information you need to choose the right one.

**Using the Status Browser**

The Status Browser is primarily a viewing tool: it displays the VSS status of each changed file, the source for each available version, and differences between, for instance, work area and database versions. The only VSS command used is the status query for each changed, added, or removed file.

Unchanged files and directories don’t show in the Status Browser. Directories and files you don’t want to see are hidden as well. Set these using the File Include Lists page.

**Using the Commit Browser**

The Commit Browser provides all of the same functions as the Status Browser. It also allows you to set the command you want to apply to each individual file, enter comments for individual files and for the whole group, then execute all of the commands with one click. JBuilder passes the commands, the comments, and the files they apply to behind the scenes. In the case of a very large checkin, having a local VSS client improves performance.

*Note* Comments for each file may be viewed from the Info page in the history pane of the JBuilder IDE. Other pages in the history pane provide views of previous revisions and allow you to edit the current buffer version.

**Version labeling**

Version labels, or tags, allow you to take a snapshot of the entire project at any point in time. Since different files change at different rates, a project of 100 files can contain 100 different current revision numbers. Version labels
allow you to mark the whole project without regard for the revision status of any individual files.

Version labels are typically used to mark milestones such as releases and other important stages in a project’s lifecycle.

JBuilder applies the tag exactly as you type it to all of the files in the active project. If necessary, tags can be altered using Visual SourceSafe.

Checking in a new project

If JBuilder doesn’t know where to find the runtime, the first step of this wizard prompts the user for that location. Once this path is set, that step of the wizard no longer appears in subsequent uses.

The usual first two steps of this wizard are Select Visual SourceSafe Database Directory and Enter Username And Password. These are also the first two steps of the Pull Project From Visual SourceSafe wizard, and are discussed in that section. Both wizards use these steps identically.

Choosing what to include in the checkin

The Select Directories And Files To Include/Exclude step controls what’s included in the checked-in project. Expand nodes that can be expanded to view their contents. Backup and output directories are excluded by default, but that can be changed. The project file is required by JBuilder to maintain shared settings, so it’s checked for inclusion and grayed out so it can’t be changed.

Choosing which files to check out

The Select Files To Keep Checked Out step tells JBuilder which files to check back out to you after the project is checked in. This just means JBuilder applies a checkout command to the selected files immediately after the checkin command has been applied to the project.

Setting the project root in the database

The Select Location In VSS Database step sets the path to the root directory in the database and names the project in VSS. You must choose an existing root directory. This wizard doesn’t create root directories for you.

The final page of the wizard displays stdout output from the commands passed to VSS.
Checking in a new project

See also

- “Configuring the database connection” on page 9-2 to learn about the first steps of this wizard when it’s first used.
- “Selecting the Visual SourceSafe database directory” on page 9-2 to learn about the Select Visual SourceSafe Database Directory step, which becomes the first step for all subsequent uses after the runtime path is set.
- “Entering user name and password” on page 9-3 to learn about the first steps of this wizard when it’s first used.
Chapter 10

Rational ClearCase in JBuilder

JBuilder’s integration of Rational ClearCase allows you to perform the most common ClearCase version control tasks from within the development environment. JBuilder simplifies some tasks, such as adding nested directories: JBuilder recursively adds the subdirectories and files you select, so you don’t have to add each one individually. JBuilder provides support for the use of both snapshot (static) and dynamic views with the base ClearCase product. This integration also supports the Unified Change Management (UCM) workflow process by associating UCM activities (new or existing) with standard version control operations, such as checkout, checkin, or add.

ClearCase commands are generally available from two places in the IDE: the Team menu from the main menu bar, and the context (right-click) menu in the project pane.

ClearCase in JBuilder is supported on Windows, Solaris, and Linux. All instructions assume that you have a compatible ClearCase client installed and configured on your machine and that you have appropriate access to a compatible ClearCase server. For a list of supported (and compatible) versions, see the JBuilder Data Sheet on the Borland web site. If you have questions about your ClearCase installation, please contact your ClearCase administrator.

Note for Linux users
The ClearCase integration is developed and tested on RedHat Linux 6.2. To use ClearCase on Linux,

1 Install the kernel patch from Rational and recompile the kernel to complete ClearCase installation on Linux.

2 Put the cleartool command in the PATH to enable ClearCase support in JBuilder.
Selecting ClearCase as your version control system

Support for Linux clients is available as of ClearCase Release 4.1; prior versions support only servers.

Selecting ClearCase as your version control system

A JBuilder project must be under ClearCase control before ClearCase commands become available. There are two ways to ensure this: pull an existing project from the repository and configure the connection automatically, or select ClearCase by hand from an open project.

To select ClearCase by hand from an open project,

1. Choose Team | Select Project VCS.

   The Select Project VCS dialog box appears:

2. Choose ClearCase from the list of version control systems in the dialog box.

3. Click OK or press Enter to close the dialog box.

   This populates the Team menu with appropriate commands and makes the ClearCase menu available from the context (right-click) menu in the project pane.

Viewing ClearCase connection properties

Once your ClearCase client is installed and configured, there is very little to configure within JBuilder to use ClearCase. The ClearCase Properties dialog box (Team | Configure ClearCase) shows the configuration of ClearCase as returned by the cleartool -version command.

The Show Console Messages checkbox enables the display of commands JBuilder issues (and corresponding responses) to perform ClearCase operations. Commands are displayed in the message pane. This feature is disabled (unchecked) by default.

The Load Checkout Comments For Checkin checkbox enables the automatic reuse of comments entered when files are checked out. When
Load Checkout Comments For Checkin is enabled (checked), the comments entered when a file is checked out will automatically be loaded into the Comment field in the ClearCase Checkin dialog box when the file is checked back in. This feature is disabled by default.
ClearCase integration is a feature of JBuilder Enterprise

JBuilder supports creating a versioned object base (VOB), mounting a VOB, checking files in and out, undoing a checkout, and labeling projects. JBuilder also provides a merge mechanism to support checkins of files that have been modified on the server since your last checkout.

Opening or creating a project for ClearCase

The simplest way to mount a ClearCase VOB, open or create a project for files in the VOB, and check the files out, is with the Project for ClearCase wizard. The Project for ClearCase wizard can be accessed from the Project page in the object gallery, or from the Team menu (Team | Project For ClearCase). To start the wizard from the Team menu, you must have a ClearCase as the selected version control system (VCS) for the active project.

From the object gallery,

   The object gallery appears.
2. Select the Project tab.
3. Select Project For ClearCase.
4. Double-click the icon, click OK, or press Enter to start the Project For ClearCase wizard.

From the Team menu, choose Team | Project For ClearCase.
Both of these ways access the Project For ClearCase wizard. This wizard provides drop-down lists for the available VOBs and views.
Using the Project For ClearCase wizard

Once the wizard appears, follow these steps:

1. Select the view type, dynamic or snapshot, using the View Type radio buttons.

2. Choose a view from the View drop-down list.
   The View drop-down list is populated with all available views, based on the type of view, dynamic or snapshot, specified with the View Type radio buttons.

3. Choose a VOB to mount from the VOB drop-down list.

4. Ensure that the path in the Directory field points to a directory in the selected view.
   If necessary, click the ellipsis (...) button, and navigate to the directory.

   **Note** The Project For ClearCase wizard prevents you from choosing the root directory of a snapshot view. For projects that contain EJB or Web applications, you must choose a directory within a VOB in the snapshot. Projects with EJB and Web applications include files that are dependant on the location of the JBuilder project (.jpx) file. If the project file is outside of a VOB, then some of the files required for EJBs and Web applications cannot be checked in to ClearCase.

5. Click OK to close the wizard.

JBuilder opens the project in the specified directory, or creates a new project if the directory does not contain one.

Checking out a file

Checking out a file makes it writable to the person who has it checked out. More than one user at a time can have the same files checked out if unreserved checkouts are used. In JBuilder, the Check Out command is available from the Team menu and from the context menu in the project pane.

To check out the active file from the Team menu,

1. Choose Team | Checkout “<filename>".
The ClearCase Checkout dialog box appears, listing the active file to check out.

2 Type a comment in the Comment field according to your usual practices.

If you enable the Load Checkout Comments For Checkin feature, JBuilder will automatically include the comments you enter here when you check the file back in.

Note If the project is using the Unified Change Management (UCM) process, you must select an existing UCM activity from the drop-down list or click New to create a new activity to associate with the checkout.

3 Select the type of checkout, Reserved or Unreserved, to perform.

By default, Reserved is selected and the Unreserved Checkout Instead If Already Reserved checkbox is checked. JBuilder will attempt to perform a reserved checkout, giving you the exclusive right to check in a new version of the file. If someone else has performed a reserved checkout of the file, JBuilder performs an unreserved checkout. An unreserved checkout does not guarantee the right to create the successor version. If you select Reserved, and the Unreserved Checkout Instead If Already Reserved checkbox is not checked, no checkout will be performed if someone else has performed a reserved checkout of the file.

4 Click OK to check out the file.

5 Click OK to close the dialog box.

Note By default, JBuilder displays dialog boxes to confirm the success or failure of version control system (VCS) operations. To configure these confirmation dialog boxes to close automatically after successful VCS operations, check Close VCS Dialogs Automatically After Successful Operation in the IDE Options dialog box (Tools | IDE Options).
Checking out a file

To check out one or more files by using the context menu,

1 Select the file or files in the project pane.
   Use the Ctrl key or the Shift key to select more than one file.
2 Right-click the selection, and choose ClearCase|Checkout Files from the context menu.
   The ClearCase Checkout dialog box appears. If you have selected multiple files, the dialog box lists the files to check out. Look at the list of files and make sure it matches what you intended.

   Note The ClearCase Checkout dialog box lists only selected files that can be checked out. Selected files that are not under ClearCase control or that are already checked out will be ignored.

3 Type a comment in the Comment field according to your usual practices.
   If you enable the Load Checkout Comments For Checkin feature, JBuilder will automatically include the comments you enter here when you check the files back in.

   Note If the project is using the Unified Change Management (UCM) process, you must select an existing UCM activity from the drop-down list or click New to create a new activity to associate with the checkout.

4 Select the type of checkout, Reserved or Unreserved, to perform.
   By default, Reserved is selected and the Unreserved Checkout Instead If Already Reserved checkbox is checked. JBuilder will attempt to perform a reserved checkout, giving you the exclusive right to check in a new version of the file. If someone else has performed a reserved checkout of the file, JBuilder performs an unreserved checkout. An unreserved checkout does not guarantee the right to create the
successor version. If you select Reserved, and the Unreserved Checkout Instead If Already Reserved checkbox is not checked, no checkout will be performed if someone else has performed a reserved checkout of the file.

5 Click OK to check out the files.

6 Click OK to close the dialog box.

**Undoing a checkout**

Undo Checkout revokes your write access and discards the changes you made to a file since you checked it out. In JBuilder, this command is available from the Team menu and from the context menu in the project pane.

To undo checkouts from the Team menu, choose Team|Undo Checkout “<filename>”.

To undo checkouts on files by using the context menu,

1 Select the file or files in the project pane.
   
   Use the Ctrl key or the Shift key to select more than one file.

2 Right-click the selection, and choose ClearCase|Undo Checkout from the context menu.

Your write access to those files is revoked and the changes you made are discarded.

**Hijacking files**

The Hijack command (Team|Hijack) is only available for use with files in snapshot views. Hijacking a file makes the file editable (changes from read-only to a read-write state) without checking the file out. This command lets you work on files when the ClearCase server is not accessible, such as when you are disconnected from the network.

Use ClearCase’s Update View wizard (Team|ClearCase Tools|Update View Wizard) to convert the file’s ClearCase status from Hijacked to checked out (Reserved checkout or Unreserved checkout) when the server does become available.

To hijack the active file from the Team menu, choose Team|Hijack “<filename>”.
Posting changes to a single file or set of files

To use the context menu to hijack one or more files,

1. Select the file or files in the project pane.
   Use the Ctrl key or the Shift key to select more than one file.
2. Right-click the selection, and choose ClearCase | Hijack Files from the context menu.
3. Click OK to hijack the files.
4. Click OK to close the dialog box.

Posting changes to a single file or set of files

The Checkin command is available from the Team menu and from the context menu in the project pane.

To check in the active file from the Team menu,

1. Choose Team | Checkin "<filename>".
   The ClearCase Checkin dialog box appears, listing the active file to check in.

2. Type a comment in the Comment field according to your usual practices.
   If you enabled the Load Checkout Comments For Checkin feature, JBuilder automatically includes the comments you entered when the file was checked out. You can edit or edit or add to these comments.
Posting changes to a single file or set of files

Note If the project is using the Unified Change Management (UCM) process, you must select an existing UCM activity from the drop-down list or click New to create a new activity to associate with the checkin.

3 Select the type of checkout to perform after the file has been checked in. By default, no checkout is performed, and the file status will change to Read Only when checkin operation is completed. If you select Reserved or Unreserved, JBuilder checks the file back out immediately after the checkin, so you can continue to work on it.

4 Click OK to check in the files.

5 Click OK to close the dialog box.

Note By default, JBuilder displays dialog boxes to confirm the success or failure of version control system (VCS) operations. To configure these confirmation dialog boxes to close automatically after successful VCS operations, check Close VCS Dialogs Automatically After Successful Operation in the IDE Options dialog box (Tools | IDE Options).

To check in files by using the context menu,

1 Select the file or files in the project pane.
   Use the Ctrl key or the Shift key to select more than one file.

2 Right-click the selection, and choose ClearCase | Checkin Files from the context menu.
   The ClearCase Checkin dialog box appears. If you have selected multiple files, the dialog box lists the files to check in. Files must be
Posting changes to a single file or set of files

checked out before they are modified and checked back in. Look at the list of files and make sure it matches what you intended.

3 Type a comment in the Comment field according to your usual practices.

If you enabled the Load Checkout Comments For Checkin feature, JBuilder automatically includes the comments you entered when the files were checked out. You can edit or edit or add to these comments.

Note If the project is using the Unified Change Management (UCM) process, you must select an existing UCM activity from the drop-down list or click New to create a new activity to associate with the checkin.

4 Select the type of checkout to perform after the file or files have been checked in.

By default, no checkout is performed, and the file status will change to Read Only when checkin operation is completed. If you select Reserved or Unreserved, JBuilder checks the files back out immediately after the checkin, so you can continue to work on them.

5 Click OK to check in the files.

6 Click OK to close the dialog box.

Merging differences between versions

When the Check In command is used, JBuilder looks for updates before checking the modified files in. If files have been changed by others since you last checked them out, those files are displayed in the Merging Files dialog box. The Merging Files dialog box asks you to choose between merging the differences and then checking in the files, or canceling the checkin on the files that have changed in both places.
Synchronizing the project file

If the Merge Files dialog box appears when you check files in, look at the files affected by the merge. If you want to merge them, click OK to merge or click Cancel to keep your local version of the files without checking them in yet.

If you click OK, JBuilder will merge the files. This is normally straightforward, and each file becomes one unified version incorporating both sets of changes.

**Important** Occasionally, there may be a *merge conflict*. This happens when there’s a difference between workspace and server versions in the same physical part of a file. JBuilder displays files with merge conflicts in another dialog box, and does not check them in. Merge conflicts must be resolved within ClearCase.

Files that don’t need merging will be checked in normally, regardless of how many other files in the list need merging.

Synchronizing the project file

Instead of automatically being committed and updated with the rest of the project, the project file must be committed and updated in a separate process. This allows you to control when and whether global settings, such as path settings or required libraries, get posted to the repository.

**Important** Complex projects may contain other projects within them. This means that one project may have several project files, at different levels within it. Only the current, top-level .jpx project file for the currently active project is handled separately. Other project files in the project are updated with the rest of the project.

To commit your local version of the project file to the repository, choose Team | Add Current Project “<project file name>”.

To update your workspace with the latest repository version of the project file, choose Team | Pull Latest Project “<project file name>”.

The project file is maintained separately from the rest of the project. There are two reasons for this:

- Changes made to the project file can include path settings and other changes that would affect the work of others who share the same project. Maintaining the project file separately allows you to make changes to files and paths and to test those changes before you need to alter the project file.

- Many people may be working on the same project, but using different files within it. If each of them commits the project file every time they commit the project, it can create difficulties for the other users of the same project.
Checking status or committing changes for the entire project

JBuilder provides browsers that can display pertinent version control information and execute commands on many files at once. Both the Status Browser and the Commit Browser query ClearCase for the status and repository version of each changed file, and combine that information with information that JBuilder already has from your work session.

Using the Status Browser

The Status Browser is primarily a viewing tool: it displays the ClearCase status of each file that has been modified in the workspace, the source for each available version, and differences between, for instance, workspace and database versions. The only ClearCase command used is the status query for each changed, added, or removed file.

Unchanged files don’t show in the Status Browser. Directories and files you don’t want to see are hidden as well. Set these using the File Include Lists page.

Browsing the project’s status

The Status Browser (Team | Status Browser) displays all the files in your project and the version control status of each one. It has two pages: the Changes page and the File Include Lists page.

In the Changes page, the directory tree view is in the left pane and the list of files is in the right. Select the Full List node to view a list of all the files in the project. Select a directory node to see a list of the files in that directory. The version control status of each file is noted in the Status column of the list.

Select a file in the file list. The tabbed pane below the file list allows you to view the source code of the selected file in any way that’s pertinent. For instance, if you choose a file that has been changed locally, the Workspace Source, Repository Source, and Workspace Diff tabs become available. If you choose a file that has been changed on the database, the Workspace Source, Repository Source, Repository Diff, and Complete Diff tabs become available.

<table>
<thead>
<tr>
<th>Source view</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workspace Source</td>
<td>This file’s source code from the current workspace version.</td>
</tr>
<tr>
<td>Repository Source</td>
<td>This file’s source code from the current database version.</td>
</tr>
<tr>
<td>Workspace Diff</td>
<td>This file’s most recent changes in your workspace.</td>
</tr>
<tr>
<td>Repository Diff</td>
<td>This file’s most recent changes in the database.</td>
</tr>
<tr>
<td>Complete Diff</td>
<td>Differences between the current version of this file in the database and the current version in your workspace.</td>
</tr>
</tbody>
</table>
Choosing files to include

The Status Browser displays and determines which files remain under version control and which files remain strictly local. It also filters files that remain under version control but are not necessarily of interest to you. It has two pages: Team Include List and Personal Include List.

The Team Include List shows which files are under version control and which are local. These files are tracked by the `<projectname>.jpx` file. The information you enter here is stored in that file as well, because this is a team-wide project setting. The files that are checked in the Team Include List are the files that everyone needs to be able to use.

The backup files (`bak` directory) and the `<projectname>.jpx.local` file are normally excluded (unchecked). Check your company’s policy about what files should be included and what should be excluded from a team project checkin.

**Caution**

The shared `.jpx` file must be included (checked in the Team Include List) to maintain version control for the project with ClearCase in JBuilder.

The Personal Include List filters your personal view, so that only the files you specify can show up in the Changes page. These files are tracked by the `<projectname>.jpx.local` file. The Personal Include List is entirely for your convenience. Since you won’t necessarily be working on every file in the project, you don’t necessarily want to look at all of them whenever you do a checkin.
Checking status or committing changes for the entire project

This simple chart illustrates the concept of available files as compared to the files needed by an individual developer:

The Personal Include List page lets you keep in view only the files that you need. The rest stay hidden until you want to see them again.

Using the Commit Browser

The Commit Browser provides all of the same functions as the Status Browser. It also allows you to set the command you want to apply to each individual file, enter comments for individual files and for the whole group, then execute all of the commands with one click. JBuilder passes the commands, the comments, and the files they apply to behind the scenes.

Note Comments for each file may be viewed on the Info page in the history pane of the JBuilder IDE. Other pages in the history pane provide views of previous revisions and allow you to edit the current buffer version.

Checking in the project

To check in changes for every modified file in the project at once, use the Commit Browser (Team | Commit Browser). The Commit Browser provides status messages and context-sensitive default ClearCase actions for each modified file in the project.

The Commit Browser also provides alternative ClearCase commands in drop-down menus in the Action column. The drop-down menus are context-sensitive; available commands vary depending on the status of the file. The default option varies depending on how the file achieved that status.
Checking status or committing changes for the entire project

The Commit Browser has three pages: Commits, Summary Comment, and File Include Lists. It opens to the Commits page by default.

An expandable directory tree view is in the left pane and the list of files is in the right. Expand the tree view to display subdirectories and directory contents.

In the Commits page, tabbed panes display the selected file's source in five ways:

<table>
<thead>
<tr>
<th>Tab</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workspace Source</td>
<td>This file's source code from the current workspace version.</td>
</tr>
<tr>
<td>Repository Source</td>
<td>This file's source code from the latest database version.</td>
</tr>
<tr>
<td>Workspace Diff</td>
<td>This file's most recent changes in your workspace.</td>
</tr>
<tr>
<td>Repository Diff</td>
<td>This file's most recent changes in the database.</td>
</tr>
<tr>
<td>Complete Diff</td>
<td>The differences between the current local version of this file and the latest database version.</td>
</tr>
</tbody>
</table>

Action options

JBuilder chooses default options to place in the Action column based on the file's status as reflected in the Status column. For changes made within JBuilder, the default actions are chosen not only according to the file's status but how it reached that status. For instance, if a file isn't in the workspace, it might be because it was removed from the project or because it has not yet been checked out. JBuilder senses the reason that it's...
Checking status or committing changes for the entire project

not in the workspace and chooses the most likely option to list as the
default: Remove From Repository or Check Out. Additions and removals
made from this browser are automatically checked in.

The list of files has a context menu that supports multiple selections. To
apply the same option to several files, select the files you want, using the
_Shift_ key or the _Control_ key to extend the selection beyond the first file
selected. Right-click any of the selected files, then choose the option you
want from the context menu.

If you want to check in some but not all changed files in the project, select
No Action as the option for the files you don’t want to check in. You can
also auto-select No Action for all of the files in a directory by right-clicking
the directory in the node tree view and choosing Perform No Action For
All In Directory from the context menu.

It’s best to add and delete files from within JBuilder, so JBuilder can
perceive the reason for the changed status of the files and select the most
appropriate default option. The following table lists common options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>Check in the change to the version control database.</td>
</tr>
<tr>
<td>Add</td>
<td>Add this file so it’s stored in the VOB.</td>
</tr>
<tr>
<td>Delete</td>
<td>This file has already been removed from the VOB; this removes it from your workspace.</td>
</tr>
<tr>
<td>Get</td>
<td>This file has already been added to the database; this checks it out to your workspace.</td>
</tr>
<tr>
<td>Revert</td>
<td>Update the workspace with the latest version of this file in the VOB, discarding all changes made since you checked the file out.</td>
</tr>
<tr>
<td>No Action</td>
<td>When you click OK, this file will not be touched by a version control operation of any kind. It will be exactly as you left it before you invoked the Commit Browser.</td>
</tr>
</tbody>
</table>

The following table demonstrates a few status notations and associated
options.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Listed status</th>
<th>Default option</th>
<th>Alternative options</th>
</tr>
</thead>
<tbody>
<tr>
<td>File changed in the workspace</td>
<td>Changed In Workspace</td>
<td>Commit to Repository</td>
<td>• Undo Checkout</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No Action</td>
</tr>
<tr>
<td>File added to version control</td>
<td>Not In Repository</td>
<td>Add to Repository</td>
<td>• Delete Locally</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No Action</td>
</tr>
<tr>
<td>File deleted from version control</td>
<td>Not In Workspace</td>
<td>If removed from within JBuilder: Remove From Repository, Get From Repository</td>
<td>• Undo Checkout</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No Action</td>
</tr>
</tbody>
</table>
Adding a new file

New files and directories often get added to the project. If these file system components are created inside a ClearCase view, they are seen as view private elements. They need to be added to the VOB so that they can be shared. The Add command in JBuilder is available from the Team menu and from the context menu in the project pane.

To add the active file from the Team menu,

1. Choose Team | Add "<filename>"

   The ClearCase Add dialog box appears, listing the active file to add.

2. Type a comment in the Comment field according to your usual practices.

   If the project is using the Unified Change Management (UCM) process, you must select an existing UCM activity from the drop-down list or click New to create a new activity to associate with the operation.
Adding a new file

3 Select the type of checkout to perform after the file has been added to ClearCase.

By default, no checkout is performed, and the file status will change to Read Only when checkin operation is completed. If you select Reserved or Unreserved, JBuilder checks the file back out immediately after the checkin, so you can continue to work on it.

4 Click OK to add the file.

5 Click OK to close the dialog box.

Note By default, JBuilder displays dialog boxes to confirm the success or failure of version control system (VCS) operations. To configure these confirmation dialog boxes to close automatically after successful VCS operations, check Close VCS Dialogs Automatically After Successful Operation in the IDE Options dialog box (Tools | IDE Options).

To add files by using the context menu,

1 Select the file or files in the project pane.

Use the Ctrl key or the Shift key to select more than one file.

2 Right-click the selection, and choose ClearCase | Add Files from the context menu.

The ClearCase Add dialog box appears. If you have selected multiple files, the dialog box lists the files to check in. Look at the list of files and make sure it matches what you intended.

3 Type a comment in the Comment field according to your usual practices.

Note If the project is using the Unified Change Management (UCM) process, you must select an existing UCM activity from the drop-down list or click New to create a new activity to associate with the checkin.

4 Select the type of checkout to perform after the file or files have been added to ClearCase.

By default, no checkout is performed, and the file status will change to Read Only when checkin operation is completed. If you select Reserved or Unreserved, JBuilder checks the files back out immediately after the checkin, so you can continue to work on them.

5 Click OK to add the files.

The files are added to the VOB and become available to other users.

6 Click OK to close the dialog box.

Note A file element can only be added if its directory tree is under ClearCase; therefore, when you add a file inside of a directory that’s not under ClearCase, JBuilder traverses up the tree until it finds a directory that is
under ClearCase. Then it adds the new directory structure under that parent directory, and adds the file you selected in the appropriate place in the tree.

Version labeling (tagging)

Tags, or version labels, allow you to take a snapshot of the entire project at any point in time. Since different files change at different rates, a project of 100 files can contain 100 different current revision numbers. Version labels let you mark the evolution of the entire project without reference to changes in individual files. Version labels are applied with the ClearCase native tools.

To apply a version label, choose Team | ClearCase Tools | Apply Label. This starts ClearCase’s Apply Label wizard. Follow the instructions in the wizard and in the ClearCase documentation to apply a label to all files in the project.
In the Rational ClearCase integration, the key task available for working on a new project is creating a new VOB.

Placing a new project into ClearCase

Once you have created a new project and chosen ClearCase as the version control system to use for it, you can place the project under ClearCase control with the Place Project Into ClearCase wizard. The active project will be imported into a specified versioned object base (VOB), and associated with a dynamic view. The VOB and dynamic view must already exist.

Note

If you have sufficient rights, you can create VOBs and views yourself. See “Working with views and VOBs” on page 12-2 for more information.

To place your project into ClearCase,

1. Choose Team | Place Project Into ClearCase.
   
   The Place Project wizard appears.

2. Choose a VOB from the VOB drop-down list.

3. Choose a dynamic view from the Dynamic View drop-down list.

4. Type in or select the path to the physical location of the VOB on your server in the Directory field.

   The directory must already exist and be under ClearCase source control, and the directory must not contain any other JBuilder projects.
5 Click OK to close the wizard.

The active project will first be imported into the VOB on the ClearCase server, and then it will be closed. Files and directories that are in the output and backup paths are not checked in, but are left as view-private. The project in the VOB will then be opened and checked out (unreserved).

Note JBuilder preserves the original workspace; therefore, after it creates the new VOB, it copies files into the VOB rather than moving them from the original workspace. All source and non-generated resource files in the project are copied into the VOB.

Working with views and VOBs

JBuilder provides menu commands for starting native ClearCase tools from within JBuilder, including ClearCase’s View Creation wizard (Team | ClearCase Tools | Create View) and VOB Creation wizard (Team | ClearCase Tools | Create VOB). If you have sufficient rights, you can create, update, or remove views and VOBs using the ClearCase Tools. Consult the ClearCase documentation for instructions and additional information.

If you do not have sufficient rights, contact your ClearCase administrator.
Team development is a feature of JBuilder SE and Enterprise

This chapter provides additional general information on version management and additional assistance with the different version control tools supported by JBuilder.

Please see “Developer support and resources” on page 1-4 for information on helpful JBuilder and Borland links and newsgroups.

The version control system has a published API. You can use it to customize an existing integration or to design your own.

To familiarize yourself with the OpenTools API,

1 Choose Help | Help Topics.
2 In the Contents page of the help viewer, open JBuilder OpenTools Documentation.
3 Start with the first chapter, “Developing OpenTools.”

The topics contained in “Developing OpenTools” explain the APIs conceptually. This information makes it much easier to understand the APIs and get the most out of them.
4 Open the next chapter, “OpenTools API Documentation”.

Read the introductory material for information on the structure and use of the API documentation.
5 Open the com.borland.primetime.teamdev packages.

These tell you how to integrate your own version control tool into the JBuilder IDE or customize the existing integrations.

There are other APIs available from Help | JBuilder OpenTools Links, but the version control API is not one of them.
General revision management resources

For comparisons and reviews of configuration management tools and version control systems, visit the CM Today Yellow Pages at [http://www.cmtoday.com/yp/configuration_management.html](http://www.cmtoday.com/yp/configuration_management.html).

Resources for supported tools

The following third-party resources provide additional information about the version control systems supported by team development.

**CVS**

- **List of newsgroups**: [http://www.cvshome.org/communication.html](http://www.cvshome.org/communication.html)
- **CVS Home**: [http://www.cvshome.org/](http://www.cvshome.org/)
- **Special considerations**:
  - Get CVS for various platforms from the source: [http://www.cvshome.org/dev/codes.html](http://www.cvshome.org/dev/codes.html)
  - CVS for the Mac (see the OpenTools documentation to extend JBuilder’s capabilities): [http://www.maccvs.org/](http://www.maccvs.org/) (home page)

**Rational ClearCase**


**Visual SourceSafe**

- **Special considerations**:
This tutorial covers the use of most of the JBuilder CVS commands. You don’t need to set anything up before you do this tutorial, once you have JBuilder installed.

For this lesson, you will create a local CVS repository to work from. Since it will be created locally by you, you will have the access you need to complete all of the steps in this tutorial.

Many version control features can only be exposed in a multi-user environment, so we’ll replicate a multi-user environment in this tutorial. You’ll be acting as two users, User One and User Two.

In this tutorial, you’ll create a project, put it into a module, and check that module out into two different directories. You’ll manipulate the same project module in both directories. This way you can replicate a multi-user development environment on one machine. This exposes you to as many features as possible within the CVS integration.

JBuilder provides extensive diff-handling features in the history pane in the content pane. To learn more about the history pane features, please see “Comparing files and versions” in Building Applications with JBuilder.

The Accessibility options section in the JBuilder Quick Tips contains tips on using JBuilder features to improve JBuilder’s ease of use for people with disabilities.

For information on documentation conventions used in this tutorial and other JBuilder documentation, see “Documentation conventions” on page 1-3.
Step 1: Setting up

In this step, you’ll create a JBuilder project and tell JBuilder that you’ll use CVS for the new project. This will provide you with everything you need to start using CVS in JBuilder.

To create a new project,


The Project wizard appears.

2. In Step 1 of the Project wizard, make the project name `thisproject`, and accept the other defaults.

Step 1 should look like this:

3. Click Finish in Step 1 to create the project.

Now you must tell JBuilder which version control system to use for this project:

1. Choose Team | Select Project VCS.

The Select Project VCS dialog box appears:
Step 2: Creating a repository and creating a module

2 Select CVS from the list of version control systems.
3 Click OK to close the dialog box.

Open the Team menu and notice that the menu has expanded, and some items on it are now available.

Step 2: Creating a repository and creating a module

Now let’s create the local repository to use for the rest of this tutorial:

1 Choose Team | Create Local Repository.
   The Create A Local CVS Repository wizard appears:

   ![Create Local CVS Repository Wizard](image)

   2 Type `localrepos` at or near the root (according to your preferred practices).
   3 Click OK.

   JBuilder creates the repository with the name `localrepos` and returns a confirmation dialog box.

   4 Click OK to close the confirmation dialog box.

   Now you’re ready to use CVS. The first thing to do is to create a module to work from. Creating a module in JBuilder involves configuring the connection to CVS and creating a CVS module to check into your repository. Once the module is in the repository, all of its contents are under CVS control and more commands for this project are enabled.

   To create a new module,

   1 Choose Team | Place Project Into CVS.

   The Place Project Into CVS wizard appears. The first step of this wizard, the CVS Module Properties step, configures this project’s connection to CVS.

   2 Under Connect Type, select Local.

   Since you don’t need to log into your machine to proceed, the Login Settings gray out.
3. Under Repository Path, type in or navigate to the `localrepos` repository you made.
Since your connection type is Local, the ellipsis (...) button is available, allowing you to navigate to your repository instead of typing the path in.

4. Name the module `thismodule`.
When you’re done, the settings will look like this:

The text at the bottom of the box displays the CVSROOT. Before you proceed, you can try changing elements in this page of the wizard to see how the CVSROOT changes to match.
Please be sure your entries match these instructions before you continue.

5. Click Next to go to Step 2, the CVS Project Description step:

Accept the default comment.
Step 2: Creating a repository and creating a module

6 Click Next to go to Step 3, the Choose Directories And Files To Include/Exclude step:

![Screenshot of Choose Directories And Files To Include/Exclude dialog box]

Notice that the directories of backup (bak) and derived (class) files are automatically excluded from the repository.

7 Accept this default and click Finish.

The Place Project Into CVS feedback dialog box appears:

![Screenshot of Place Project Into CVS feedback dialog box]

This feedback dialog box displays the stdout output from CVS as it comes back and lets you know when the task succeeds.

8 Click Close to close the dialog box.
Step 3: Adding files and committing file changes

You have created a CVS module for your new project, and placed your project into the CVS repository. Now you’ll add files, make changes, and commit them. This way, your work is stored in the repository, where it’s under version control and available to other users.

In this step, you will perform the following tasks:

• Add the files to your JBuilder project
• Add the files to the CVS repository
• Modify files and commit the changes

The changes used in this tutorial are extremely simple, so you can focus on the process rather than the code.

Adding files to your JBuilder project

First, create a group of files to work with:

1 Choose File | New to open the object gallery.
2 Select Application from the object gallery’s General page.
3 Click OK to start the Application wizard.
4 Accept all the defaults in Step 1 and click Next.
5 Check all of the checkboxes in the Options area in Step 2:
Step 3: Adding files and committing file changes

6 Click Finish to close the wizard.

The Application wizard creates seven new files and the thisproject package, and returns you to the AppBrowser. Note the added files:

Now you have several different types of files to work with for the rest of this tutorial.

Adding the files to the CVS repository

Add the new files to CVS:

1. Select about.png in the project pane.

2. Hold down the Shift key and click openfile.png to select about.png, openfile.png, and all of the files in between.

If you accidentally selected the HTML file, hold down the Ctrl key, and click the file in the project pane. This will deselect the HTML file but leave the other files selected.
Step 3: Adding files and committing file changes

3 Right-click the selection, and choose CVS|Add Files from the context menu.

The commands on the menu apply to all of the selected files. Available commands vary depending on what types of files are selected and which edition of JBuilder you’re using.

The CVS Add dialog box appears:

Note Putting files under version control in CVS is normally a two step process. First, you run the `cvs add` command to tell CVS you want to version control the files, and then you run `cvs commit` to confirm the addition and check the files into the repository. JBuilder’s Add Files command performs both steps.
Step 3: Adding files and committing file changes

4 Enter the following comment, and click OK.

   Adding default files from the Application wizard.

   JBuilder shows a dialog box that confirms the addition of your files to CVS.

5 Click OK to close the dialog box.

   Note CVS can only add a file that exists within the workspace. If you ever want to add a file that’s outside the project directory, copy it into the project directory, add it to the JBuilder project once it’s inside the directory, and then add it to CVS.

Modifying files and committing the changes

Now change a file and commit the change. Committing a change means posting the modified version of the file to the repository, so that the changes you made are available to other users.

Make the change in Frame1.java, which is already active in the content pane:

1 Insert // first change below the first line of code, as the following graphic illustrates:
Step 4: Checking out an existing module

In the previous steps, you created a module for your project, added files to CVS, and manipulated a file in CVS. In this step, you will assume the role of another developer, User Two, and check out and work with the existing module/project under version control.

A module is created once, but it needs to be checked out by every user who works on the project. Therefore, you will check out an existing module and make changes to the project, just as if you were using a module that someone else had created.

In this step, you will perform the following tasks:

- Check out a module/project from the CVS repository
- Modify a file in the checked out project
- Remove a file from the checked out project
- Commit the changes to the repository

Checking out a module from the repository

Use the object gallery to check a module out of the CVS repository quickly:

1. Choose File | New to open the object gallery.
2. Click the Project tab, and select Pull Project From CVS.

2 Choose Team | Commit “Frame1.java”.
   This opens the CVS Commit dialog box.

3 Type the following comment in the space provided in the CVS Commit dialog box, and click OK.
   Made first change.

4 When the commit operation is completed, click OK to close the dialog box.

5 Save the project (File | Save Project “thisproject.jpx”).

6 Close the project, thisproject, by clicking the Close Project icon in the project toolbar.
Step 4: Checking out an existing module

3 Click OK to start the Pull Project From CVS wizard. You will use this wizard to check out the module you created before. Now that you’re playing the part of a second developer on the same project, however, you’ll use a different directory as a workspace.

4 Leave the default path to jbproject intact but change the name of the target directory to dir2:

5 Click Next to go to the CVS Module Properties step.

6 Set your connection type to Local.

7 Click Next to go to the CVS Repository Properties page:

JBuilder keeps track of the repositories, modules, and branches you use so you can select them from drop-down lists in the appropriate areas.
Step 4: Checking out an existing module

instead of typing them in each time you create or check out a new module.

Accept the default repository settings and Autosave Files Before CVS Operations.

8 Click Finish.

The Checking Out CVS Project feedback dialog box displays the CVS process of putting the thisproject module into dir2.

9 When the dialog box indicates that the checkout is complete, click Close to close the dialog box.

Modifying a file in the checked out project

As User Two, you have checked out the module you created as User One. Now make a change to it.

Tip If you want to check which directory you’re in and you don’t have a window title bar that shows it, choose Project | Project Properties and look at the Paths page. It will show you the entire path of the active project.

1 Expand the thisproject package node in the project pane.

2 Double-click Frame1.java in the project pane to open it in the editor.

3 Change // first change to // second change.

You don’t have to commit immediately. You often won’t when you’re busy.

Removing a file from the checked out project

Remove a file from the project now,

1 Double-click the file Frame1_AboutBox.java in the project pane to open it in the content pane.

Once the file is in the content pane, the file-level commands on the Team menu will apply to it.
Step 4: Checking out an existing module

2 Choose Team | Remove “Frame1_AboutBox.java”:

![Image of the CVS Remove dialog box]

The CVS Remove dialog box will appear, prompting you for a comment.

3 Type in, Removing Frame1_AboutBox.java in the CVS Remove dialog box, and click OK.

![Image of the confirmation message]

Read the confirmation message when it appears:

Removing a file from the CVS repository is normally a three-step process. First, the file is removed from the local workspace, then the *cvs remove* command is run, and then the *cvs commit* command is executed. The JBuilder Remove command performs all steps required to remove the file from the workspace and the CVS repository.

4 Click OK to close the CVS Remove dialog box.
Step 5: Updating a project

Committing the changes to the repository

Now commit the change you made before:
1 Select Frame1.java in the project pane.
2 Right-click the selected file and choose CVS|Commit “Frame1.java” from the context menu.
   This opens the CVS Commit dialog box.
3 Type the following comment in the space provided in the CVS Commit dialog box, and click OK.
   Second change made.
4 When the commit operation is completed, click OK to close the dialog box.
5 Save and close the project.

Step 5: Updating a project

You have created a new module and checked out an existing one. Next, as User One, update a project to retrieve changes from the repository.

Updating keeps your workspace current with changes other developers may have made to the files you’re using. This is our first opportunity in this tutorial to be exposed to others’ changes, so we’ll start doing this now.

First you’ll view a file’s status in relation to CVS. You can do that at any time.
1 Choose File|Reopen and choose <home>/jbproject/thisproject/thisproject.jpx.
2 Open Frame1.java in the content pane.
3 Choose Team|Status For “Frame1.java” to open the CVS Status dialog box.
   The CVS Status dialog box should indicate that the file has changed in the repository. This was the change you made as User Two in the previous step of this tutorial.
4 Click OK to close the CVS Status dialog box.
Step 5: Updating a project

Update your project to synchronize your workspace with what’s in the repository:

1. Choose Team | Update Project.
2. Click OK in the Update Project dialog box.
3. Click OK to close the dialog box.

Notice that JBuilder provides a project update in the message pane, telling you exactly what was changed in your workspace by this update:

Tip

It is a good idea to update the files in your workspace (Team | Update Project) periodically to reconcile your work with any revisions committed to the repository since your last checkout or update. This merges repository changes into your workspace, and can help uncover potential conflicts before you commit changes. If there’s still a conflict when you commit, JBuilder engages its merge conflict management features.

4. Open the Window menu.

This menu has options that control the AppBrowser’s appearance and provide access to all open projects and to all open files within the active project. Projects are assigned letters and files are assigned numbers when they are displayed in this menu. This provides a way of distinguishing between projects or files that have the same names.
Step 6: Simulating multiple users

Notice that the current project file, thisproject.jpx, is given the letter A in the menu:

Step 6: Simulating multiple users

Now let’s simulate a situation you might encounter in the course of a real work session. In this scenario, you work on several files at a time, and make a number of changes. You commit the changes you make to files periodically, but not every time you make a change. It may be more productive to make several changes to a file, without interruption, before committing the changes to the repository, instead of committing every change individually. Sometimes, however, someone else is working on the same files at the same time and makes changes you don’t know about.

In this step, you will perform the following tasks:

- Modify and commit files as User Two
- Modify and save (not commit) Frame1.java as User One
- Modify and commit Frame1.java as User Two

Modify and commit files as User Two

First, we’ll set up the work User Two does on the same files User One works on:

2. Choose Team | Update Project.

The Update Project dialog box appears. Remember that you must maintain your project’s CVS status regularly. Updating before a work session is good practice.
Step 6: Simulating multiple users

3 Click OK to start the update.

Because the files in this project match the files in the repository, there should be no changes to your workspace, and, consequently, no CVS project update messages in the message pane.

4 When the update operation has completed, click OK to close the Update Project dialog box.

5 Expand the thisproject package in the project pane to reveal the files inside.

6 Double-click Application1.java in the project pane to open it in the content pane.

7 Type // third change under the first line of code:

8 Choose Team | Commit “Application1.java”.

This opens the CVS Commit dialog box.

9 Type the following comment in the space provided in the CVS Commit dialog box, and click OK.

Third change made in this project.

10 When the commit operation is completed, click OK to close the dialog box.
Step 6: Simulating multiple users

Let’s make another change as User Two:

1. Select the Frame1.java file tab in the content pane.
2. Set the @author Javadoc tag value to H. Marx.

Locate the following line:

```java
* @author not attributable
```

Replace `not attributable` with `H. Marx`:

```java
* @author H. Marx
```

Note that there’s a row:column notation in the status bar of the content pane. The Author entry is at row 12.

3. Choose Team | Update “Frame1.java”.
4. Click OK to start the update, and when the update is complete, click OK to close the dialog box.
5. Choose Team | Commit “Frame1.java”.
   This opens the CVS Commit dialog box.
6. Type the following comment in the space provided in the CVS Commit dialog box, and click OK.
   
   Fourth change: H. Marx as author.
7. When the commit operation is completed, click OK to close the dialog box.
Step 7: Using the Status Browser

Modify and save (not commit) Frame1.java as User One

Make a change as User One to the initial project, <home>/jbproject/thisproject/thisproject.jpx. The purpose of this change is to make the Workspace Diff feature available later on in the tutorial.

1. Select the Frame1.java file tab in the content pane.
2. Type // fifth change in row 6 of the file. (Look in the status bar for your precise location in the code.)
3. Save the file (File|Save “Frame1.java”) but don’t commit it.

Modify and commit Frame1.java as User Two

Now go back to being User Two and create the other half of a conflict in the project, <home>/jbproject/dir2/thisproject.jpx:

1. Open the Window menu, and choose B thisproject.jpx.
   There are two project files called thisproject.jpx. Remember that the one that has the letter A in front of it is the one used by User One. This means the one with the letter B is the one used by User Two. As you move your mouse pointer over the project file name in the Window menu, the full path for the project is displayed in the status bar.
2. Open User Two’s version of Frame1.java.
3. In Frame1.java, go to row 6 and type // conflicts with the fifth change in the same row, row 6.
4. Choose Team|Commit “Frame1.java”.
   This opens the CVS Commit dialog box.
5. Type the following comment in the space provided in the CVS Commit dialog box, and click OK.
   Conflict created.
6. When the commit operation is completed, click OK to close the dialog box.

Step 7: Using the Status Browser

The Status Browser is a sophisticated viewing tool that provides an overview of the status of your files and a quick look at the source code of each changed file. For the rest of this tutorial, you will work as User One.
Step 7: Using the Status Browser

To view the version control status for files,

1 Choose Window | A thisproject.jspx.
2 Choose Team | Status Browser.

The Status Browser appears. The Full List node at the top of the tree is selected by default:

3 Select Application1.java, and note which file tabs are available in the source view pane:

- **Workspace Source** displays what’s in the buffer. The buffer is the most recent local version on your disk; it includes unsaved changes.
- **Repository Source** displays the code in the module.
- **Repository Diff** displays the differences between the repository version you’re working from and the most recent version in the repository, allowing you to see the changes others have made while you were working on the file.

4 Select Frame1.java, which has conflicts.

All of the file tabs are available:

- **Workspace Diff** displays the differences between the workspace version you started with and the most recent version you’ve created, allowing you to see all of the changes you made in this work session.

Note that this dialog box displays only changed files. Files that have not been changed don’t show here.
Step 7: Using the Status Browser

Complete Diff displays the differences between the latest repository version and the most current workspace version of the selected file. Merge conflicts are fully displayed here.

5 Select the Complete Diff tab:

Notice how merge conflicts are tagged. JBuilder uses vivid colors and gutter markers to distinguish the different parts of a merge conflict.

Navigate from one diff block (area of differences between versions) and another by using the arrows at the left end of the status bar. Notice the information provided in the status bar.

6 Go ahead and view the diffs (blocks of different text in the same area of code) on each tab, if you want to.

7 Click Close to close the dialog box.

The Status Browser doesn’t execute any CVS commands. The only changes it makes are to the personal and shared file lists. The Commit Browser, however, can be used to execute the most commonly used CVS commands, such as cvs update, cvs commit, and cvs remove. In the next step of the tutorial, you will use the Commit Browser to update files in your workspace, and update files in the repository.
Using the Commit Browser to manage files in the version control system

The Commit Browser offers the same viewing features as the Status Browser, and, more importantly, it allows you to apply CVS processes to all changed files in any way you want to. You will use the Commit Browser to,

- View file status
- Update your workspace
- Locate and resolve merge conflicts
- Enter comments and commit changes

Viewing version control status with the Commit Browser

To view the status of files in the project with the Commit Browser,

1. Choose Team | Commit Browser.
   The Commit Browser appears.
2. Select Frame1.java from the file list:
Step 8: Using the Commit Browser

Look at the file list in this image. Notice that the popup Action menu for this file is activated and that the default is Update. Updating is a good way to start addressing a merge conflict.

Each file’s available actions depend on the state the file is in—whether it was changed in repository, is not in the workspace, has conflicts, etc. The action JBuilder chooses as the default depends on how it reached that state. For instance, a file that’s not in the workspace may have been either added to the repository or deleted in the workspace. If it was added to the repository, JBuilder suggests updating it. If it was deleted in the workspace, JBuilder suggests committing it.

The Commit Browser has the same source view tabs as the Status Browser. Notice the extra tab for an individual comment. Below that pane, notice the Use Summary Comment checkbox. The Summary Comment can be applied to all of the files being committed. If the Use Summary Comment box on this page is checked, only the comment you enter under the Summary Comment tab will be used for the file that’s selected in the file list.

Updating the workspace with the Commit Browser

Next, look at the File Include Lists page:

1. Select the File Include Lists page.

2. Select the Personal Include List page, and expand the `src` and `thisproject` nodes:
These checkboxes mark the files and directories that can be seen and acted on in the commit browser. Certain files may not be pertinent to you. Remove them from your personal view of the project by unchecking them here. It may not make sense to put some files under version control; keep them out of CVS by unchecking them here.

3 Leave the defaults in both the Team and Personal Include lists. Return to the Commits page.

4 Right-click Frame1.java, and choose the option to update the file individually from the context menu.

Notice that, once the file’s CVS status has been resolved, it disappears from the Commit Browser.

**Locating and resolving merge conflicts with the Commit Browser**

Time to finish this task up and take care of that conflict:

1 Click Commit to commit the remaining file, Application1.java.

The dialog box displays the progress of the commit operation.

2 Click Close to close the dialog box.

There is a CVS Project Updates warning in the message pane about the merge conflict.

3 Click the warning in the message pane.

This displays the Merge Conflicts page in the history pane. The conflicting areas in the workspace source and repository source are highlighted in separate panes on the Merge Conflicts page. Radio buttons in the gutters of the workspace and repository panes are used to select the changes to keep.

Lines for a conflict that has been selected to be kept are highlighted in yellow, and are further indicated by plus (+) signs in the gutter. Changes to be discarded are highlighted in red and have minus (-) signs in the gutter. The preview pane at the bottom of the Merge conflicts page shows what the file source will look like when the changes are applied. By default, the changes in the workspace are selected to be retained.

You may close the message pane by right-clicking in it and choosing Remove “CVS Project Updates” Tab from the context menu.

4 Click the radio button in the gutter next to the highlighted conflict in the repository source.

The highlight colors change, and a plus (+) sign appears in the gutter of the selected line in the repository pane. The preview pane shows how
Step 8: Using the Commit Browser

the file will look if the changes are applied. The icon in the gutter of the preview pane indicates the change is from the repository:

5 Click Apply to apply the changes.

This updates the file in your workspace to match the file in the repository.

Note Applying changes updates the editor buffer for the file in the workspace. Until you commit your changes, you can use the undo command (Edit | Undo or Ctrl+z) to restore the conflicts. The undo command is not available from the history pane, so you will need to switch to a different view.

6 Commit the file, using either the Team menu or the project pane context menu.
Entering comments and committing changes with the Commit Browser

The Commit Browser can, of course, also be used to commit changes you make to files in your project. When you commit changes with the Commit Browser, you can take advantage of its comment features. The file changes we will make will be trivial, and the comments will not be relevant to version control, but they will illustrate how JBuilder structures its comment features.

To enter comments and commit changes with the Commit Browser,

1. Double-click Application1.java in the project pane to open it in the content pane.
2. Change the second line of code from // third change to // final change.
3. Choose Team | Commit Browser to open the Commit Browser.
4. Select Application1.java.
5. In the Individual Comment area, type equals four.

Notice that when you start to type, the Summary Comment box unchecks itself. In the Commits page, the Summary Comment option is an either/or option: you can choose either an individual comment or the summary comment.

However, you can use both individual and summary comments from the Summary Comment page.
Step 8: Using the Commit Browser

6 Select the Summary Comment page.
7 Enter the first half of the statement:

Two plus two . . .

Notice that the Prepend Summary Comment checkbox stays checked.

You can’t see both comments together from the Commit Browser. We’ll use the Info page in the history pane of the AppBrowser to view complete comments, but commit the changes to Application1.java first so the comments are visible in the Info page. For information on the history pane, please see “Comparing files and versions” in Building Applications with JBuilder.

8 Click Commit to update the repository with the changes you made to Application1.java.
9 When the commit operation is completed, click Close to close the Version Control Operations dialog box.

Let’s go back and look at those comments. As you recall, we first entered an individual comment, equals four. Then we entered a summary comment, Two plus two . . . . Let’s see how JBuilder puts them together.

1 Select Application1.java in the content pane.
2 Select the History tab at the bottom of the content pane.
Step 8: Using the Commit Browser

3 Select the Info page. This provides the fullest view of comments, especially as you can resize the Comments pane.

4 Select revision number 1.3 and look at the Comments area. Notice how JBuilder puts the summary comment first and separates the summary comment from the individual comment with an extra line break.

We’re done. You may close the project.

Congratulations! You have successfully used nearly all of the JBuilder CVS integration features that you’d normally use in daily work.
Index

B
Borland
  contacting 1-4
  developer support 1-4
e-mail 1-6
newsgroups 1-5
online resources 1-5
reporting bugs 1-6
technical support 1-4
World Wide Web 1-5

C
ClearCase integration See version control, ClearCase
Commit Browser
  in ClearCase 11-12
  in CVS 3-7
  in VSS 7-15
console messages
  enabling for ClearCase 10-2
CVS integration See version control, CVS
CVS tutorial 14-1

D
documentation conventions 1-3
  platform conventions 1-4

E
environment variables 5-8

F
file types
  associating 5-7
fonts
  JBuilder documentation conventions 1-3

G
glossary
  CVS terms 2-2

M
merge conflicts
  reconciling in CVS 3-11
  reconciling in VSS 7-8
modules, creating in CVS 4-1

N
newsgroups
  Borland 1-5
  public 1-6

R
revision management See version control

S
selecting version control systems
  CVS 4-1
setting environment variables 5-8
Status Browser
  in ClearCase 11-10
  in CVS 3-6
  in VSS 7-13

T
Team Development See version control

U
Usenet newsgroups 1-6

V
version control 1-1
  how it works 1-2
  resources and bookmarks 13-1
version control, ClearCase 10-1
  adding new files 11-15
  applying tags 11-17
  checkin comments 10-2
  checking in files 11-6
  checking in the entire project 11-12
  checking out 11-2
  Commit Browser, Action options 11-13
  Commit Browser, Summary comments 11-15
  creating a new VOB 12-1
  File Include lists 11-11
  menus 10-2
  merging workspace and server versions 11-8
  mounting a VOB/creating a project 11-1
  Project For ClearCase wizard 11-2
  showing console messages 10-2
  source views 11-13
  Status Browser 11-10
  undo checkout 11-5
version control, CVS 3-1
adding files 3-16
checking file status 3-18
checking out modules 3-2
Commit Browser Action options 3-9
committing files 3-5
committing the project 3-7
configuring connection 2-1
creating local repositories 4-4
CVSROOT syntax 5-8
environment variables 5-8
File Include lists 3-6
glossary 2-2
handling binary files 5-7
managing file status 3-18
modules, creating 4-1
overview 2-1
programmer’s guide to integration 5-1
programmer’s guide, project checkin 5-5
programmer’s guide, pull project 5-3
programmer’s guide, pulling project file 5-4
pulling and posting the project file 3-15, 11-9
reconciling conflicts automatically 3-12
reconciling conflicts manually 3-14
reconciling merge conflicts 3-11
removing files 3-17
selecting CVS 4-1

Status Browser dialog 3-6
updating files 3-11
updating the project 3-15
using watches and edits 3-18
version labelling 3-17
watches and edits 3-18
version control, VSS 6-1
adding files 7-12
checking in files 7-6
checking in the entire project 7-15
checking out a project 7-1
checking out files 7-3
Commit Browser, Action options 7-16
Commit Browser, Summary comments 7-18
configuring project connection 6-1
creating a version label 7-18
File Include lists 7-14
placing project into database 8-1
posting .jpx files 7-7
programmer’s guide 9-1
pulling .jpx files 7-5
reconciling conflicts automatically 7-9
reconciling conflicts manually 7-10
reconciling merge conflicts 7-8
removing files 7-13
source views 7-16
Status Browser 7-13
undoing a file checkout 7-4
views in ClearCase See version control, ClearCase
Visual SourceSafe integration See version control, VSS
VSS integration See version control, VSS