

# Managing Application Development in JBuilder®

## JBuilder® 2005

**Borland®**  
Excellence Endures™

Borland Software Corporation  
100 Enterprise Way  
Scotts Valley, California 95066-3249  
[www.borland.com](http://www.borland.com)

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 © 1997–2004 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.

JB2005teamdev 8E8R0804  
0405060708-9 8 7 6 5 4 3 2 1  
PDF

# Contents

<b>Chapter 1</b>		
<b>Introduction</b>	<b>1</b>	
Using version control systems . . . . .	2	
Finding information and commands quickly. . . . .	2	
Documentation conventions . . . . .	3	
Developer support and resources. . . . .	4	
Contacting Borland Developer Support . . . . .	4	
Online resources. . . . .	4	
World Wide Web. . . . .	5	
Borland newsgroups. . . . .	5	
Usenet newsgroups . . . . .	5	
Reporting bugs . . . . .	5	
<b>Chapter 2</b>		
<b>Handling patches</b>	<b>7</b>	
Applying patches . . . . .	7	
Options for applying patches . . . . .	8	
Creating patches . . . . .	9	
<b>Part I</b>		
<b>Using StarTeam with JBuilder</b>		
<b>Chapter 3</b>		
<b>StarTeam in JBuilder</b>	<b>13</b>	
Finding information and commands quickly. . . . .	13	
How StarTeam integrates with JBuilder. . . . .	14	
Object gallery . . . . .	15	
Team menu . . . . .	15	
Administrative commands . . . . .	15	
File-level version control commands . . . . .	16	
Project-level version control commands . . . . .	17	
StarTeam Repository node and project view window . . . . .	18	
Project view window: folder tree pane . . . . .	18	
Project view window: upper pane. . . . .	19	
Project view window: lower pane . . . . .	19	
StarTeam Repository page toolbar . . . . .	19	
StarTeam context menus . . . . .	20	
StarTeam in the project pane. . . . .	21	
StarTeam context menus in the project view window. . . . .	22	
Context menus in the StarTeam pane . . . . .	22	
StarTeam pane. . . . .	22	
Limitations of the StarTeam integration . . . . .	22	
<b>Chapter 4</b>		
<b>Managing projects in StarTeam</b>	<b>25</b>	
Selecting StarTeam as your version control system. . . . .	25	
Placing a new project into StarTeam . . . . .	26	
Pulling an existing project from StarTeam . . . . .	29	
Changes and commands across the project . . . . .	31	
Using the Changes and Commits pages . . . . .	31	
Tree view . . . . .	32	
Table of changed files . . . . .	32	
Summary Comment . . . . .	34	
Tabbed source views . . . . .	34	
Using the StarTeam page . . . . .	35	
File Include Lists . . . . .	37	
Synchronizing project settings. . . . .	38	
<b>Chapter 5</b>		
<b>Managing files in StarTeam</b>	<b>39</b>	
Adding files . . . . .	39	
Removing files . . . . .	41	
Checking in files . . . . .	42	
Checking out files . . . . .	44	
Moving files . . . . .	46	
Merge conflicts. . . . .	46	
Reconciling merge conflicts . . . . .	47	
Reconciling conflicts manually . . . . .	47	
Comparing file revisions . . . . .	48	
Locking and unlocking files in StarTeam. . . . .	49	
Finding files in StarTeam . . . . .	50	
<b>Chapter 6</b>		
<b>Working with process items and topics in StarTeam</b>	<b>51</b>	
Adding, linking, and removing process items . . . . .	52	
Editing process items . . . . .	52	
Setting the active process item . . . . .	52	
<b>Chapter 7</b>		
<b>Performing StarTeam administrative tasks</b>	<b>55</b>	
Personal Options . . . . .	55	
Viewing StarTeam connection properties . . . . .	56	
Logging on and off. . . . .	56	
Creating and changing views . . . . .	56	
Using labels . . . . .	57	
Managing shortcuts . . . . .	59	
Creating process item shortcuts . . . . .	59	
<b>Chapter 8</b>		
<b>Version control reference: StarTeam</b>	<b>61</b>	
Version control system API . . . . .	61	
General revision management resources . . . . .	61	
StarTeam resources . . . . .	61	

## Part II

### Using CVS with JBuilder

---

#### Chapter 9

#### **CVS in JBuilder 65**

Finding information and commands quickly . . . . .	66
CVS glossary . . . . .	66

#### Chapter 10

#### **Working on an existing project in CVS 69**

Checking out an existing project . . . . .	70
Posting file changes . . . . .	72
Changes and commands across the project . . . . .	73
Using the Changes and Commits pages . . . . .	74
Tree view . . . . .	74
Table of changed files . . . . .	74
Summary Comment . . . . .	76
Tabbed source views . . . . .	76
File Include Lists . . . . .	77
Synchronizing with changes in the repository . . . . .	78
Updating a single file . . . . .	78
Reverting a file back to its checked out state . . . . .	78
Reconciling CVS merge conflicts . . . . .	78
Reconciling conflicts manually . . . . .	79
Updating projects or project groups . . . . .	80
Synchronizing project settings . . . . .	80
Adding files . . . . .	81
Removing files . . . . .	82
Working with labels and branches . . . . .	83
Using version labels . . . . .	83
Creating a branch . . . . .	84
Special updates and merges . . . . .	84
Tracking file status in CVS . . . . .	86
Checking a file's CVS status . . . . .	86
File access notification . . . . .	87
CVS Watches . . . . .	87

#### Chapter 11

#### **Working on a new project in CVS 89**

Placing a new project into CVS . . . . .	89
Checking in a project for the first time . . . . .	90
Creating local repositories . . . . .	91

#### Chapter 12

#### **Programmer's guide to the CVS integration 93**

Guide to the CVS integration . . . . .	93
Using CVS as your version control system . . . . .	94
Selecting CVS . . . . .	94
Configuring the repository connection . . . . .	94
Creating a local repository . . . . .	94
Getting material out of the repository . . . . .	94
Pulling a project . . . . .	94
Pulling or posting a project file . . . . .	95
Checking out files . . . . .	95
Updating files . . . . .	95
Removing files . . . . .	95

Getting material into the repository . . . . .	96
Checking in a project . . . . .	96
Adding new files . . . . .	96
Committing changes . . . . .	96
Managing the module . . . . .	97
Version labeling . . . . .	97
Using watches . . . . .	97
CVS reference . . . . .	97
Handling binary files in CVS . . . . .	97
Checking and setting user environment variables . . . . .	98
Linux and Solaris . . . . .	98
Windows XP . . . . .	98
Windows 2000 . . . . .	98
Using CVS with SSH . . . . .	99
Configuring the SSH connection in JBuilder . . . . .	99

#### Chapter 13

#### **Version control reference: CVS 101**

Version control system API . . . . .	101
General revision management resources . . . . .	101
CVS resources . . . . .	102

## Part III

### Using Subversion with JBuilder

---

#### Chapter 14

#### **Subversion in JBuilder 105**

Finding information and commands quickly . . . . .	106
--	-----

#### Chapter 15

#### **Working on an existing project in Subversion 107**

Checking out an existing project . . . . .	107
Posting file changes . . . . .	108
Changes and commands across the project . . . . .	109
Using the Changes and Commits pages . . . . .	110
Tree view . . . . .	110
Table of changed files . . . . .	110
Summary Comment . . . . .	112
Tabbed source views . . . . .	112
File Include Lists . . . . .	113
Synchronizing with changes in the repository . . . . .	114
Updating a single file . . . . .	114
Reverting a file back to its checked out state . . . . .	114
Reconciling Subversion merge conflicts . . . . .	114
Resolving conflicts . . . . .	115
Updating the project . . . . .	115
Synchronizing project settings . . . . .	115
Adding files . . . . .	116
Removing files . . . . .	117

#### Chapter 16

#### **Working on a new project in Subversion 119**

Placing a new project into Subversion . . . . .	119
Checking in a project for the first time . . . . .	120
Creating local repositories . . . . .	121

## Part IV

### Using Visual SourceSafe with JBuilder

---

#### Chapter 17

##### Visual SourceSafe in JBuilder 125

Configuring the connection . . . . .	125
Finding information and commands quickly. . . . .	126

#### Chapter 18

##### Working on an existing project in VSS 127

Pulling an existing project. . . . .	127
Selecting the database directory . . . . .	128
Entering security information . . . . .	128
Selecting a project. . . . .	129
Choosing a target directory . . . . .	129
Checking out files . . . . .	129
Undoing a checkout . . . . .	130
Checking in files. . . . .	130
Synchronizing project settings. . . . .	131
Reconciling Visual SourceSafe merge conflicts . . . . .	132
Reconciling conflicts manually. . . . .	133
Adding and removing files . . . . .	133
Adding files . . . . .	133
Removing files . . . . .	134
Changes and commands across the project . . . . .	135
Using the Changes and Commits pages . . . . .	135
Tree view . . . . .	136
Table of changed files . . . . .	136
Summary Comment . . . . .	138
Tabbed source views . . . . .	138
File Include Lists. . . . .	139
Version labeling a project . . . . .	140

#### Chapter 19

##### Working on a new project in VSS 141

Placing a new project into Visual SourceSafe . . . . .	141
--	-----

#### Chapter 20

##### Programmer's guide to the VSS integration 145

Configuring the connection . . . . .	145
Selecting VSS . . . . .	145
Configuring the database connection . . . . .	145
Runtime location and performance. . . . .	146
Pulling an existing project. . . . .	146
Selecting the Visual SourceSafe database directory . . . . .	146
Entering user name and password . . . . .	146
Selecting a Visual SourceSafe project. . . . .	147
Selecting an empty target directory . . . . .	147
Checking out files . . . . .	147
Undoing a checkout . . . . .	147
Checking in files. . . . .	147
Checking the project file in or out . . . . .	148

Adding and removing files . . . . .	148
Adding files . . . . .	148
Removing files . . . . .	148
Checking in the entire project . . . . .	148
Using the Status Browser . . . . .	149
Using the Commit Browser . . . . .	149
Version labeling . . . . .	149
Checking in a new project . . . . .	149
Choosing what to include in the checkin . . . . .	150
Choosing which files to check out . . . . .	150
Setting the project root in the database . . . . .	150

#### Chapter 21

##### Version control reference: Visual SourceSafe 151

Version control system API . . . . .	151
General revision management resources . . . . .	152
Visual SourceSafe resources . . . . .	152

## Part V

### Using ClearCase with JBuilder

---

#### Chapter 22

##### ClearCase in JBuilder 155

Selecting ClearCase as your version control system . . . . .	155
Viewing ClearCase connection properties . . . . .	156
Finding information and commands quickly . . . . .	157

#### Chapter 23

##### Working on an existing project in ClearCase 159

Opening or creating a project for ClearCase . . . . .	159
Using the Project For ClearCase wizard . . . . .	160
Checking out a file . . . . .	160
Undoing a checkout . . . . .	161
Hijacking files . . . . .	162
Posting changes to a single file or set of files . . . . .	162
Merging differences between versions . . . . .	163
Synchronizing project settings . . . . .	164
Changes and commands across the project . . . . .	164
Using the Changes and Commits pages . . . . .	165
Tree view . . . . .	166
Table of changed files . . . . .	166
Summary Comment. . . . .	167
Tabbed source views . . . . .	168
File Include Lists . . . . .	168
Adding a new file. . . . .	169
Version labeling (tagging) . . . . .	170

#### Chapter 24

##### Working on a new project in ClearCase 171

Placing a new project into ClearCase . . . . .	171
Working with views and VOBs. . . . .	172

Chapter 25	
<b>Version control reference:</b>	
<b>ClearCase</b>	<b>173</b>
Version control system API . . . . .	173
General revision management resources . . . . .	173
ClearCase resources . . . . .	174

## Part VI

### Requirements Management

---

Chapter 26	
<b>Managing requirements using</b>	
<b>CaliberRM</b>	<b>177</b>
Configuring a CaliberRM connection . . . . .	178
Connecting to a CaliberRM server . . . . .	178
Opening a CaliberRM connection . . . . .	179
Switching between CaliberRM connections . . . . .	180
Changing the CaliberRM project and	
baseline . . . . .	180
Troubleshooting the CaliberRM connection . . . . .	180

The CaliberRM UI . . . . .	181
The toolbar . . . . .	181
The Table view . . . . .	182
Displaying requirements in the Table view . . . . .	182
Icons in the Table view . . . . .	183
The Details panel . . . . .	183
Description tab . . . . .	183
User Defined Attributes tab . . . . .	183
Comment Traces tab . . . . .	183
Configuring filters . . . . .	184
Adding requirements to code . . . . .	186
Adding a requirement as a comment . . . . .	186
Updating source comments . . . . .	187

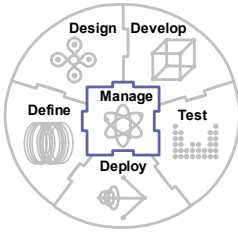
<b>Index</b>	<b>189</b>
--------------	------------

# Tables

1.1	Typeface and symbol conventions . . . . .	3	15.2	Subversion Commit Browser: Version control options . . . . .	111
1.2	Platform conventions . . . . .	4	15.3	Subversion: Source views . . . . .	112
2.1	Common options used when applying patches . . . . .	8	18.1	VSS: Project-wide version control status, files, and commands . . . . .	135
3.1	StarTeam Repository page toolbar commands . . . . .	19	18.2	VSS Commit Browser: Version control options . . . . .	136
4.1	StarTeam: Project-wide version control status, files, and commands . . . . .	31	18.3	VSS: Source views . . . . .	138
4.2	StarTeam Commit Browser: Version control options . . . . .	32	23.1	ClearCase: Project-wide version control status, files, and commands . . . . .	164
4.3	StarTeam: Source views . . . . .	34	23.2	ClearCase Commit Browser: Version control options . . . . .	166
10.1	CVS: Project-wide version control status, files, and commands . . . . .	73	23.3	ClearCase: Source views . . . . .	168
10.2	CVS Commit Browser: Version control options . . . . .	75	26.1	CaliberRM message boxes . . . . .	180
10.3	CVS: Source views . . . . .	76	26.2	Toolbar buttons . . . . .	181
15.1	Subversion: Project-wide version control status, files, and commands . . . . .	109	26.3	Table view icons . . . . .	183
			26.4	Comment Traces tab toolbar buttons . . . . .	184







# Introduction

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 software project, and version labeling allows you to take a snapshot of the entire file set at any stage of development.

**CVS and Subversion integrations are in all editions of JBuilder**

**StarTeam, Visual SourceSafe, and ClearCase integrations are features of JBuilder Developer and Enterprise**

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 provides interfaces with StarTeam, Concurrent Versions System (CVS), Subversion, Visual SourceSafe, and 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 also provides a plug-in for the CaliberRM requirements management system. The CaliberRM plug-in allows you to view requirements, link requirements to source code files, insert requirements into source code, and determine if requirements on the server have changed.

**Note** The Version Control OpenTool and JBuilder's open, extensible architecture allow you to integrate other version control systems, as well. For more information, see the OpenTools documentation.

## 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.

## Finding information and commands quickly

---

JBuilder provides access to relevant version control information and commands from several convenient places, supporting many different working styles.

The history view in the content pane lets you examine prior revisions, look at diffs, read revision lists and comments, view checkins line by line, and handle merge conflicts if they arise.

In the project pane and the file tabs, JBuilder decorates the icons of files changed in version control. It provides textual notations and tool tips in the project pane describing each changed file's state. It checks the repository state every 15 minutes by default. Customize these settings in the Tools|Preferences dialog box, on the Project tab's Decorations page.

Pertinent version control commands are available in these places:

- Choose the Team menu on the main menu bar for a complete list of commands.
- In the project pane, right-click these nodes:
  - Project node, in the Project tab
  - File node, in the Project tab
  - Any directory that's part of the project, in the File tab
- To pull a project from the repository, choose File|New|Project.
- To update the entire project recursively, type `jbuilder -update <projectname.jpx>` at the command line.

**Note** The update command is the only version control command available from the command line. It executes the project VCS's update command against every file in the project.

# 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

Typeface	Meaning
<b>Bold</b>	Bold is used for java tools, bmj (Borland Make for Java), bcj (Borland Compiler for Java), and compiler options. For example: <b>javac</b> , <b>bmj</b> , <b>-classpath</b> .
<i>Italics</i>	Italicized words are used for new terms being defined, for book titles, and occasionally for emphasis.
<i>Keycaps</i>	This typeface indicates a key on your keyboard, such as "Press <i>Esc</i> to exit a menu."
Monospaced type	Monospaced type represents the following: <ul style="list-style-type: none"> <li>■ text as it appears onscreen</li> <li>■ anything you must type, such as "Type <code>Hello World</code> in the Title field of the Application wizard."</li> <li>■ file names</li> <li>■ path names</li> <li>■ directory and folder names</li> <li>■ commands, such as <code>SET PATH</code></li> <li>■ Java code</li> <li>■ Java data types, such as <code>boolean</code>, <code>int</code>, and <code>long</code>.</li> <li>■ Java identifiers, such as names of variables, classes, package names, interfaces, components, properties, methods, and events</li> <li>■ argument names</li> <li>■ field names</li> <li>■ Java keywords, such as <code>void</code> and <code>static</code></li> </ul>
[ ]	Square brackets in text or syntax listings enclose optional items. Do not type the brackets.
< >	<p>Angle brackets are used to indicate variables in directory paths, command options, and code samples. JDK 5.0 uses angle brackets to denote generics.</p> <p>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.</p> <p>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.</p> <p>See "Using command-line tools" in <i>Building Applications with JBuilder</i> for more information.</p> <p><b>Note:</b> 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.</p>
<i>Italics, serif</i>	This formatting is used to indicate variable strings within code samples that are already using angle brackets as delimiters. For example, <code>&lt;url="jdbc:borland:jbuilder\\samples\\guestbook.jds"&gt;</code>
...	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.

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

Item	Meaning
Paths	Directory paths in the documentation are indicated with a forward slash (/). For Windows platforms, use a backslash (\).
Home directory	The location of the standard home directory varies by platform and is indicated with a variable, <home>. <ul style="list-style-type: none"> <li>■ For UNIX, Linux, and OS X, the home directory can vary. For example, it could be /user/&lt;username&gt; or /home/&lt;username&gt;</li> <li>■ For Windows NT, the home directory is C:\Winnt\Profiles\&lt;username&gt;</li> <li>■ For Windows 2000 and XP, the home directory is C:\Documents and Settings\&lt;username&gt;</li> </ul>
Screen shots	Screen shots reflect the Borland Look & Feel on various platforms.

## 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 Developer Support

---

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

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:

- |                               |  |
|-------------------------------|--|
| <b>World Wide Web</b>         | <a href="http://www.borland.com/">http://www.borland.com/</a><br><a href="http://info.borland.com/techpubs/jbuilder/">http://info.borland.com/techpubs/jbuilder/</a>                           |
| <b>Electronic newsletters</b> | To subscribe to electronic newsletters, use the online form at:<br><a href="http://www.borland.com/products/newsletters/index.html">http://www.borland.com/products/newsletters/index.html</a> |

## World Wide Web

---

Check the JBuilder page of the Borland website, [www.borland.com/jbuilder](http://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://info.borland.com/techpubs/jbuilder/> (updated documentation and other files)
- <http://bdn.borland.com/> (contains our web-based news magazine for developers)

## Borland newsgroups

---

When you register JBuilder you can 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.

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 to Borland at one of the following sites:

- Support Programs page at <http://www.borland.com/devsupport/namerica/>. Click the Information link under “Reporting Defects” to open the Welcome page of Quality Central, Borland’s bug-tracking tool.
- Quality Central at <http://qc.borland.com>. Follow the instructions on the Quality Central page in the “Bugs Report” section.
- Quality Central menu command on the main Tools menu of JBuilder (Tools|Quality Central). Follow the instructions to create your QC user account and report the bug. See the Borland Quality Central documentation for more information.

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 [jpgpubs@borland.com](mailto:jpgpubs@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.

## Handling patches

Many projects (commonly, open source projects) require people to contribute by *change sets*. These change sets, or *patches*, are a formalized way of applying broad groups of changes to projects. They're also a useful way to share changes with specific colleagues when you need to work on something together but are not yet ready to commit a set of changes to the team repository.

JBuilder uses an open source tool called Patch which allows you to post patches remotely. It works by using a text file (generated by JBuilder) containing the diffs between the current version of the project in your workspace and the version you checked out of the repository, and pasting the changes into the necessary files. The diff file includes context information (generally three lines of unchanged code above and below each diff) so that Patch can paste the changes appropriately.

These commands are available on the Team menu regardless of the VCS used. Apply Patch is always available (even for projects not under version control), and Create Patch is available when your project is under version control.

### See also

- The Patch home page at <http://www.gnu.org/directory/GNU/patch.html>
- Patch documentation at <http://www.rt.com/man/patch.1.html>

## Applying patches

---

This command applies a patch to the active project in your workspace. The Patch utility copies its changes into your local versions of those files that are changed by the patch. It provides options that allow you to tune this behavior. Enter these in the Additional Options field of the Apply Patch wizard.

To apply a patch to the current project,

- 1 Choose Team|Apply Patch.

The Apply Patch wizard appears.

- 2 Enter or browse to the Patch File you want to apply.

**3 Choose a Strip Value.**

This tells Patch how many directory levels to strip from the front of the fully-qualified path name. Each increment represents one slash, or directory level.

The default value is 0, for two reasons:

- Java relies heavily on directory paths, so JBuilder preserves as much of them as possible unless you tell it otherwise.
- If you created the patch using Team!Create Patch, this is probably the correct value to use.

**4 Enter any Additional Options you want to pass when applying this patch.****5 Click OK to apply the patch and close the wizard.**

**Note** If you're working on Windows, be sure that the files needing to be patched are not in use by any program. This includes stopping JBuilder processes that use these files (such as running), but it's fine to have the files open in the IDE.

## Options for applying patches

---

Patch supports a variety of options to use when applying a patch your project. These options allow you to tune how the patch is applied and what happens with the affected files.

As with many commandline utilities, Patch's commands can be invoked using either words or short mnemonics. Both are shown below.

**Table 2.1** Common options used when applying patches

Option	Description
-b --backup	Make backup files: rename or copy all changed originals instead of removing them. The related command <code>--backup-if-mismatch</code> is used by default.
--dry-run	Display the results of applying these patches, but without changing any files.
-f --force	Apply the patch even if it seems wrong. This option is risky, and is recommended only for advanced users.
-F <number> --fuzz=<number>	Ignore up to this many lines when looking for a place to paste in a chunk of diff. The default is 2.
-l --ignore-whitespace	Tells Patch not to try to match tabs or end-of-line spaces. Alphanumeric characters must still match precisely.
-R --reverse	Reverses a patch by assuming that it was created with the old and new files switched. This command is tricky, recommended only for those familiar with Patch.

### See also

- <http://www.rt.com/man/patch.1.html> for thorough documentation of Patch and its options.



## Creating patches

---

This command creates a file indicating the diffs between your current working versions of the files in the project and the versions that you pulled from the repository.

To create a patch file for others to apply,

- 1 Type in or browse to the location of the Patch File you want to create.
- 2 Decide whether you want to Include New And Deleted Files In The Patch.

This option is checked by default.

Including new and deleted files ensures that those who apply this patch are using the correct set of files after applying the patch.

- 3 Tell JBuilder whether it should Open The Patch File In The Editor After Creating It.

This lets you immediately view the newly created patch file in the editor.



P a r t I

# Using StarTeam with JBuilder



## StarTeam in JBuilder

**StarTeam integration  
is a feature of  
JBuilder Developer  
and Enterprise**

The StarTeam integration provides access to the most critical StarTeam features and functions so you can perform version control and configuration management tasks from within JBuilder. The integration connects to the StarTeam Server using the TCP/IP (Sockets) protocol. The integration also provides some JBuilder-specific features to allow you to easily check in and check out JBuilder project source files and manage your work more easily.

**Note** The StarTeam integration for JBuilder supports StarTeam 5.4 and 6.0 Servers.

The StarTeam integration incorporates large portions of the StarTeam Cross-Platform Client into the JBuilder development environment, and the StarTeam Cross-Platform Client itself can be launched from within JBuilder.

The function and use of the StarTeam Cross-Platform Client and the elements incorporated into JBuilder are documented in detail in the *StarTeam User's Guide* and the *StarTeam Administrator's Guide*. StarTeam is a powerful tool, with comprehensive version control features and capabilities. We strongly recommend that you familiarize yourself with the StarTeam documentation before using this integration. All StarTeam documentation is available for download from the Borland web site at:

<http://info.borland.com/techpubs/starteam/>.

**Important** A StarTeam Server is required to use the StarTeam integration with JBuilder. Borland offers a free personal edition and a 30 day unlimited evaluation edition of StarTeam Server for your use with JBuilder. For more information and download links, please visit the Borland StarTeam web site at: [http://www.borland.com/products/downloads/download\\_starteam.html](http://www.borland.com/products/downloads/download_starteam.html).

### Finding information and commands quickly

---

JBuilder provides access to relevant version control information and commands from several convenient places, supporting many different working styles.

The history view in the content pane lets you examine prior revisions, look at diffs, read revision lists and comments, view checkins line by line, and handle merge conflicts if they arise.

In the project pane and the file tabs, JBuilder decorates the icons of files changed in version control. It provides textual notations and tool tips in the project pane describing each changed file's state. It checks the repository state every 15 minutes by default.

Customize these settings in the Tools|Preferences dialog box, on the Project tab's Decorations page.

Pertinent version control commands are available in these places:

- Choose the Team menu on the main menu bar for a complete list of commands.
- In the project pane, right-click these nodes:
  - Project node, in the Project tab
  - File node, in the Project tab
  - Any directory that's part of the project, in the File tab
- To pull a project from the repository, choose File|New|Project.
- To update the entire project or project group recursively and without merging, type `jbuilder -update <projectname.jpx>` at the command line.

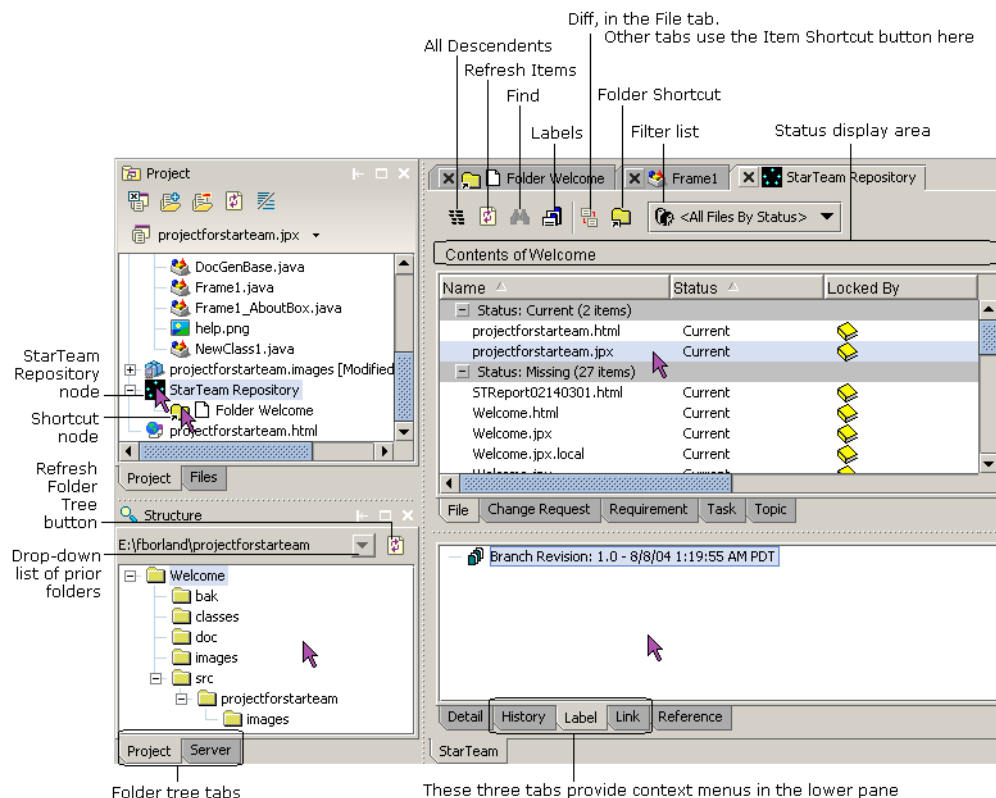
**Note** The update command is the only version control command available from the command line. It executes the project StarTeam's update command against every file in the project.

## How StarTeam integrates with JBuilder

The integration provides access to StarTeam features and functions in the following areas:

- Object gallery
- Team menu
- StarTeam Repository node and project view window
- Context menus
- StarTeam pane

The image below provides an overview of StarTeam elements in JBuilder.





In this image, the purple arrows show which areas contain context menus. To access some of StarTeam's extensive menus, right-click on the nodes or in the regions indicated by these arrows.

## Object gallery

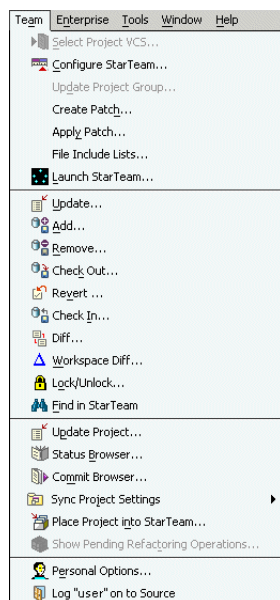


The Project page of the object gallery contains a Pull Project From StarTeam Server icon. Double-clicking the icon starts a wizard that sets the project's connection to StarTeam, and pulls (checks out) a JBuilder project from the StarTeam Server into your work area.

## Team menu

The Team menu provides access to three categories of commands for managing StarTeam and your project:

- Administrative commands
- File-level version control commands
- Project-level version control commands



Most StarTeam commands on the Team menu are enabled regardless of whether you're logged on to StarTeam. If you are not logged on to StarTeam, and you execute a command that requires you to be logged on, the Log On To StarTeam dialog box opens automatically.

### Administrative commands

The administrative commands perform actions that have an effect outside the active project, outside the IDE, or outside the StarTeam integration.

For instance, Update Project Group executes the Update command not only against the active project, but also against every project in the open project group, and it does so regardless of which version control systems the other projects use.

You can perform basic administrative tasks with the following commands:

- **Configure StarTeam** (Team|Configure StarTeam): opens a dialog box showing the StarTeam connection properties for the project.
- **Update Project Group** (Team|Update Project Group): recursively updates all projects in the active project group. Updates each project within that project's

designated version control system. This menu item is only available when the active project is part of a project group.

- **Create Patch** (Team|Create Patch): creates a patch based on the differences between your workspace versions of the files in the project and the versions you checked out from the repository. This is explained more fully in [Chapter 2, “Handling patches.”](#)
- **Apply Patch** (Team|Apply Patch): applies a patch to the active project. This is explained more fully in [Chapter 2, “Handling patches.”](#)
- **File Include Lists** (Team|File Include Lists): opens the File Include Lists dialog, allowing you to change which files are to be included in your local perspective on the project or checked into the repository. This is explained more fully in the topic, [“File Include Lists” on page 37.](#)
- **Launch StarTeam** (Team|Launch StarTeam): opens the StarTeam Cross-Platform Client. The client opens the StarTeam project associated with the active JBuilder project, and selects the project’s root folder. Only one client instance can be launched from within JBuilder. If the client is already open, the Launch StarTeam command will switch focus to the client, and open the StarTeam project associated with the active JBuilder project if it is not already open.

Although most of the features and functions of the Cross-Platform Client are available from within JBuilder, the client provides some additional or enhanced features, including Server Administration, audit logs, and the ability to create and manage StarTeam projects and views. The Cross-Platform Client also provides a time-saving toolbar filled with useful command buttons.

- **Personal Options** (Team|Personal Options): StarTeam allows team members to set personal options that suit individual work styles. These options are for the currently logged-on user on a given workstation. This is explained more fully in the topic, [“Personal Options” on page 55.](#)
- **Log “<user>” Off <ServerName>/Log “<user>” On To <ServerName>** (Team|Log Off, Team|Log On): lets you log off of the StarTeam Server to free a connection to the server, or to log on as a different user. If no default user is defined, then the menu reads Log On to <ServerName>. If you open a StarTeam object (such as a process item shortcut) or run a command that requires access to the StarTeam Server, StarTeam automatically opens the Log On dialog box.

**Note** If you have a floating license for StarTeam (sometimes called a concurrent license), there is usually a limit to the number of concurrent connections allowed to the StarTeam Server. You can log off (Team|Log “<user>” Off <ServerName>) to free a connection for another team member. If you have opened the StarTeam Cross-Platform Client (Team|Launch StarTeam), you must also close the client in addition to logging off.

## File-level version control commands

The Team menu provides access to the most common StarTeam version control commands, including:

**Note** File-level version control commands are enabled only when a file is open and selected.

- **Update** (Team|Update): updates the active file with any changes from the repository version and merges the differences between them.
- **Add** (Team|Add): adds the active file to the StarTeam repository.
- **Remove** (Team|Remove): removes the active file from the StarTeam repository.
- **Check Out** (Team|Check Out): checks the active file out from the StarTeam repository for editing.
- **Revert** (Team|Revert): discards any changes in the editor buffer for the active file, and reverts it back to the most recent checked out version.



- **Check In** (Team|Check In): checks the active file into the StarTeam repository.
- **Diff** (Team|Diff): runs StarTeam's Visual Diff tool, which shows the differences (if any) between the working file and the latest checked-in version. Under some circumstances, the latest checked in version may contain changes made by other team members.
- **Workspace Diff** (Team|Workspace Diff): runs StarTeam's Visual Diff tool, which shows the differences (if any) between your working file and the version that was checked out to the workspace.
- **Lock/Unlock** (Team|Lock/Unlock): opens the Set My Lock Status dialog box, which enables you to change the lock status of the active file (or break someone else's lock if you have the necessary privileges).
- **Find "<File>" In StarTeam** (Team|Find "<File>" In StarTeam): opens the StarTeam Repository node in the project pane, sets the folder to the folder containing this file, and selects the file in the list display.

## Project-level version control commands

The Team menu also provides access to the following JBuilder-specific commands for managing projects with StarTeam:

- **Update Project** (Team|Update Project): Synchronizes all of the files in your workspace with the repository versions, merging differences between them. Any files added to the repository will be added to your workspace, and files removed from the repository will be removed from your workspace.
- **Status Browser** (Team|Status Browser): opens the Status Browser. The Status Browser is primarily a viewing tool. It displays the version control status of each file changed either in the workspace or the repository (or both), the source for each available version, and differences between any two versions of a changed file.
- **Commit Browser** (Team|Commit Browser): opens the Commit Browser. The Commit Browser provides the viewing capabilities of the Status Browser, and also provides access to common version control operations for files that have changed. With the Commit Browser, you can set the version control command you want to apply to each changed file, enter comments for individual files and for the whole group, then execute all of the commands with one click. When a file under version control is renamed in the workspace, the Commit Browser provides an option to rename the file in the repository, or revert the file in the workspace back to the original name.
- **Sync Project Settings** (Team|Sync Project Settings): provides access to the following commands:
  - Pull Latest "<project file>"
  - Post Current "<project file>"

If the project file has not yet been added to the repository, this command is Add Current "<project file>" instead

The JBuilder project (.jpx) file is handled separately from the rest of the project in StarTeam. The Pull Latest "<project file>" command updates the project file settings for the active project in your workspace with the settings from the project file in the repository. The Post Current "<project file>" command updates the project file settings for the project file in the repository with the settings from the workspace project file.

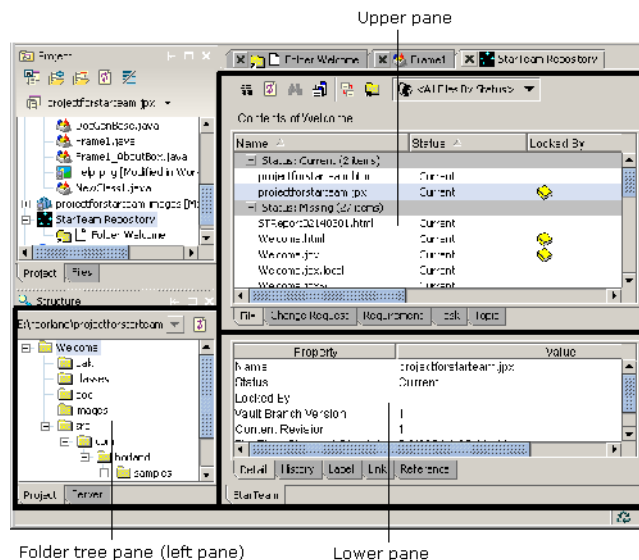
- **Place Project Into StarTeam** (Team|Place Project Into StarTeam): opens the Placing Project Into StarTeam wizard. The wizard lets you specify which files to add. The wizard also lets you specify a new or existing StarTeam server, project, view, and folder into which the files will be placed.
- **Show Pending Refactoring Operations** (Team|Show Pending Refactoring Operations): opens a StarTeam Refactorings page in the message pane which displays all refactoring events, such as moving or renaming classes, that have

occurred in the project in your workspace but have not yet been committed to the repository. If a prior attempt to commit the changes associated with a refactoring event has failed, the refactoring event in the StarTeam Refactorings page appears in red. Right-clicking one or more selected events opens a context menu that lets you remove or commit the refactoring event.

## StarTeam Repository node and project view window

When your project is under StarTeam control, a StarTeam Repository node appears in the project pane. This node lets you open the StarTeam project view window elements within JBuilder to provide extensive access to StarTeam features and information. As shown in the following figure, the project view window has three panes: a folder tree pane, an upper pane, and a lower pane. The three panes are outlined in this figure.

**Figure 3.1** StarTeam project view window panes in JBuilder



### See also

- “Looking at the Main Window” in *StarTeam User’s Guide*

### Project view window: folder tree pane

The left pane, beneath the project pane, has tabs which display the different possible hierarchies that the active view, shortcut, folder on the server, or file resides in. Choose the active view, shortcut, folder, or file by double-clicking its node in the JBuilder project pane (upper left).

As you navigate the folder hierarchy, the contents of the upper pane of the StarTeam project view window displays the items associated with the selected folder in the current tab’s directory structure, represented here in tree form.

- |               |  |
|---------------|--|
| Project tree  | With the project as root, displays the hierarchy from there down.  |
| Shortcut tree | Uses as root the root of what the shortcut points to. Displays the tree with the shortcut’s node selected.       |
| View tree     | With the view’s root as root of the display, shows the hierarchy from there down.                                |
| Server tree   | Displays everything as seen by the StarTeam server.<br>Also, lets you create a new folder from its context menu. |

**Note** The View tree is always available. If no other tabs are relevant to what appears on the right-hand side, all tabs disappear and the left pane displays the View folder hierarchy alone.

The top of the pane has a toolbar hosting useful tools:

- A drop-down list of paths to the folders you've previously selected in the current tab of this pane. Choose a path from the drop-down list to go to that StarTeam folder in the current hierarchy. The most recently selected folder sorts to the top.
- A Refresh Folder Tree button. Click this button to update the information in the current tree.

### Project view window: upper pane

The upper pane of the project view window appears in the StarTeam Repository page in the content pane when you open the StarTeam Repository node. The upper pane displays data associated with the selected folder. The type of data depends on the tab selected from the upper pane. StarTeam has a tab for each component: File, Change Request, Requirement, Task, and Topic. Double-click files or items in the upper pane to open them for editing.

**Note** Audit components are not visible within JBuilder. To view an audit log for a project, launch the StarTeam Cross-Platform Client (TeamLaunch StarTeam). See [“Limitations of the StarTeam integration” on page 22](#) for more information about the features supported by the StarTeam integration for JBuilder.

In the preceding figure, each row in the upper pane provides information about a file. If the Change Request tab had been selected, each row would have provided information about a change request. Each row is said to represent an item; that item might be a file, change request, requirement, task, or topic.

**Note** Requirements, tasks, and topics can be displayed in the upper pane in either tree or list format. Files and change requests are always displayed in lists.

#### See also

- “Viewing the Upper Pane” in *StarTeam User's Guide*

### Project view window: lower pane

The lower pane of the project view window appears in the StarTeam Repository page in the content pane when you open the StarTeam Repository node. The lower pane displays more information about the item selected from the upper pane. The type of information depends on the tab selected from the lower pane.



#### See also

- “Viewing the Lower Pane” in *StarTeam User's Guide*










### StarTeam Repository page toolbar

The toolbar at the top of the StarTeam Repository page provides button commands that affect the display of items in the upper pane. Many of these buttons are unique to the JBuilder integration. Some of the buttons apply to a specific items, and are only available when you select the tab for that type of item.

**Table 3.1** StarTeam Repository page toolbar commands

Button	Name	Description
	Refresh Items	Refreshes the status of the items displayed in the upper pane.
	Find “<item>” In StarTeam	Locates the item selected in the project pane, and displays it in the upper pane of the project view pane.

**Table 3.1** StarTeam Repository page toolbar commands (continued)

Button	Name	Description
	All Descendants	Indicates the depth for which StarTeam displays information in the upper pane of the project view window. You can also turn All Descendants on and off by selecting it from the File, Change Request, Requirement, Task, or Topic menus. When not selected, StarTeam displays information for the selected folder only. When selected (depressed), StarTeam displays information for the selected folder, its children, its children's children, and so on.
	Labels	Opens the Labels dialog box, which lets you add, delete, freeze, or edit view or revision labels.
	Difference	Opens the Visual Diff tool to compare the current version of the selected file in the workspace and the latest revision of the file in the repository.
	Create Shortcut (change request)	Creates a shortcut for each selected change request, and adds it as a subnode of the StarTeam Repository node in the project pane.
	Create Shortcut (requirement)	Creates a shortcut for each selected requirement, and adds it as a subnode of the StarTeam Repository node in the project pane.
	Create Shortcut (task)	Creates a shortcut for each selected task, and adds it as a subnode of the StarTeam Repository node in the project pane.
	Create Shortcut (topic)	Creates a shortcut for each selected topic, and adds it as a subnode of the StarTeam Repository node in the project pane.
	List View	Displays the requirements, tasks, or topics in a list format. The list can be sorted by any field shown.
	Tree View	Displays requirements, tasks, or topics in a tree. Nodes can be expanded or collapsed to display or hide subnodes.

The toolbar also contains the file filter drop-down list box. A filter is a named arrangement of data, which consists of a set of fields (for column headers), sorting and grouping information, and a query. Using filters, you could, for example, display particular groups of files, such as Files to Check Out.

The status area beneath the toolbar displays the path of the current folder, shortcut, or item.

### See also

- The “Toolbars” topic in “Looking at the Main Window” in *StarTeam User's Guide*

## StarTeam context menus

The StarTeam Integration for JBuilder includes many context menus to allow you to quickly carry out version control operations. Context menus are opened by right-clicking an object or in an area. StarTeam context menus are available when you right-click in

- The project pane
- The three panes that make up the project view window
- In the StarTeam page for a given file

**Note** StarTeam context menus are exhaustively documented in StarTeam. Start with the file mentioned below to read all about them.

Anything the JBuilder integration adds to these commands is mentioned in each pane's description in this documentation.

### See also

- “Looking at StarTeam Menus” in *StarTeam User's Guide*

## StarTeam in the project pane

The StarTeam context menu for files in the project pane lets you perform the following StarTeam commands on one or more selected files:

- Update
- Add
- Remove
- Check Out
- Check In
- Diff
- Find
- Lock/Unlock

When you right-click the JBuilder project (.jpx) file, the StarTeam context menu includes the following commands:

- Update Project
- Status Browser
- Commit Browser
- Sync Project Settings|Pull Latest "<project file>"
- Sync Project Settings|Post Current "<project file>"
- Place Project Into StarTeam Server
- Show Pending Refactoring Operations
- Personal Options
- Log "<user>" Off StarTeam Server/Log "<user>" On To StarTeam Server

These menu commands are described in ["Team menu" on page 15](#).

The context menu for item shortcuts (for a change request, requirement, task, or topic) lets you

- Open the shortcut in the project view window
- Rename item shortcuts (though not the item itself)
- Remove the item shortcut
- Set or clear the active process item
- Find the shortcut's "home" in StarTeam, opening the StarTeam Repository in the project view window
- View and edit the shortcut's properties

When you set an item to be a new process item, a temporary shortcut is created so you can quickly bring focus to the associated property editor as you work on your project. Unless you save new process items, or rename the shortcut, the shortcut will be deleted when you close the property editor for the associated process item.

The shortcut created for the active process item appears in the project pane with the gears icon added to the component type's icon.

- If you already had a shortcut to this item when it was made the active process item, then the gears are added to the shortcut icon.
- If the active process item was not a shortcut at the time it was set, then, when the item is no longer the active process item, it's removed from the list of shortcuts; in this case, it's temporary.
- If you rename the active process item shortcut, it will not be removed once it's no longer the active process item.

**Tip** JBuilder keeps you constantly notified of version control status via the Project tab in the project pane. Look here for decorators on the icons and textual annotations for each node, indicating changes in version control status and StarTeam server states. This makes it easier for you to track which files you changed since your last commit. Customize these settings in Tools|Preferences, Project tab, Decorations page.

## StarTeam context menus in the project view window

Each pane of the StarTeam project view window in JBuilder has context menus. The menus vary from pane to pane, and, in the upper and lower panes, vary with the selected page or area. For example, right-clicking in the File page in the upper pane opens a context menu different from the context menu that opens when you right-click a selected item in the pane. The top section of the upper pane's context menu is the most likely to change.

The commands on these menus are identical to the commands in the associated panes and pages in the StarTeam client, with the exceptions noted in the note below.

**Note** The context menus for the upper pane of the project view window in JBuilder do not include the Tools commands, ItemFinder and Move CR's. These commands are available only in the StarTeam client (TeamLaunch StarTeam), and only when you have StarTeam Enterprise Advantage. See [“Limitations of the StarTeam integration” on page 22](#) for more information about the features supported by the StarTeam integration for JBuilder.

StarTeam context menus are exhaustively documented in StarTeam. Start with the file mentioned below to read all about them.

### See also

- “Looking at StarTeam Menus” in *StarTeam User's Guide*

## Context menus in the StarTeam pane

The context menus are available for selected items in the History, Label, and Link pages of the StarTeam pane for the active file. The commands on these menus are identical to the commands in the lower pane of the project view window in the StarTeam client. Please refer to the *StarTeam User's Guide* for detailed documentation on using these menus.

### See also

- “Looking at StarTeam Menus” in *StarTeam User's Guide*

## StarTeam pane

---

When you work with files in JBuilder that are under StarTeam control, an additional file view tab, StarTeam, provides access to information and commands pertaining to StarTeam version control and configuration management.

The StarTeam pane provides the same information as the bottom pane of the project view window, but for the active file only.

## Limitations of the StarTeam integration

---

Although the StarTeam integration for JBuilder includes the StarTeam Cross-Platform Client, the following StarTeam features are not supported by the integration:

- **Charts:** the ability to create charts in StarTeam to view StarTeam data is supported only in the StarTeam Windows client, not in the Cross-Platform Client provided with JBuilder. With the StarTeam Windows client, you can generate a number of different charts based on the data displayed in the upper pane.
- **Virtual Meeting:** StarTeam's Virtual Meeting is supported only in the StarTeam Windows client, not in the Cross-Platform Client provided with JBuilder. The StarTeam Windows client allows you to host virtual meetings, locally or remotely, with other team members. You can share your applications with other participants and vice versa.
- **Command-line interface:** the StarTeam client provided with the JBuilder integration can only be started from within JBuilder. There are no provisions for using StarTeam's command-line interface with the StarTeam integration for JBuilder.

**Note** With the exception of the aforementioned features, the features supported by your StarTeam installation are supported by the JBuilder integration. For example, if you have StarTeam Standard, which does not support tasks, requirements, or alternate property editors (APEs), the StarTeam integration for JBuilder will not support tasks, requirements, or APEs. If you have StarTeam Enterprise, which supports tasks, your StarTeam integration will support tasks. If you have StarTeam Enterprise Advantage, your StarTeam integration will support tasks, requirements, and APEs. For more information about StarTeam, including a feature matrix, see the StarTeam product page on the Borland web site at <http://www.borland.com/starteam/index.html>.





## Managing projects in StarTeam

**StarTeam integration  
is a feature of  
JBuilder Developer  
and Enterprise**

JBuilder supports common StarTeam tasks and provides features that make applying commands to multiple files much easier. Using the StarTeam integration, you can perform the following project management tasks:

- Select StarTeam as your version control system
- Place a new project into StarTeam
- Pull an existing project from StarTeam
- Check version control status with the Status Browser
- Perform version control operations with the Commit Browser
- Update project settings

JBuilder supports different strategies for executing commands across all files in a project. Use

- The Status Browser to view all changed files
- The Commit Browser to execute commands on multiple files in the project
- Team!Update Project to update the entire project
- `jbuilder -update <projectname.jpx>` at the command line to update the entire project recursively without merging

**Note** The StarTeam Server must be running, your project must be under StarTeam control, and you must have appropriate access rights to perform the version control tasks described in the following sections.

### Selecting StarTeam as your version control system

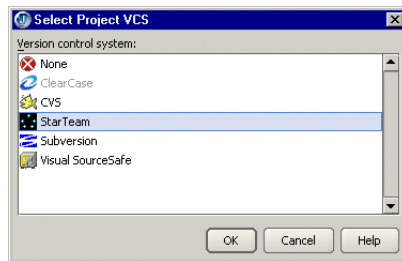
---

You must select StarTeam as the version control system (VCS) for a JBuilder project before StarTeam commands become available. Once you select StarTeam as the VCS, you can place the project under StarTeam control.

To select StarTeam as the VCS for an open project,

- 1 Choose Team|Select Project VCS.

The Select Project VCS dialog box appears:



- 2 Choose StarTeam 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 StarTeam menu available from the context (right-click) menu in the project pane. Now you can place a project into StarTeam.

## Placing a new project into StarTeam

Once you have selected StarTeam as the version control system for a project, you can place the project into a StarTeam database with the Place Project Into StarTeam command on the Team menu, or by right-clicking the project (.jpx) file in the project pane and choosing StarTeam|Place Project Into StarTeam from the context menu. Placing a project into a StarTeam database enables version control of that project and makes it accessible to other users of the version control system.

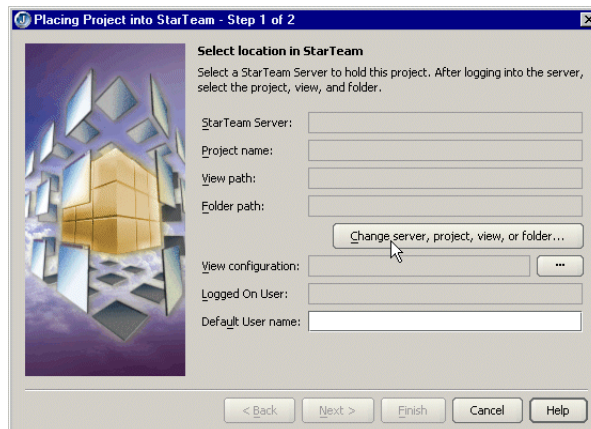
The Place Project Into StarTeam command invokes the Placing Project Into StarTeam wizard. With this wizard, you take an open JBuilder project and place it in a corresponding StarTeam project on an existing StarTeam Server. In the wizard, you specify the StarTeam Server, project, view, and folder for storing the files, and you specify which files and subdirectories to include in the StarTeam project.

To place a project into StarTeam,

- 1 Choose Team|Place Project Into StarTeam.

The Placing Project Into StarTeam wizard opens and displays the Select Location In StarTeam page.

- 2 Click the Change Server, Project, View, Or Folder button.



The Select A Root Folder dialog box appears, showing a tree view which can be expanded to display available servers, projects, views, and folders.

- 3 Expand the nodes to navigate down to the project you want to open.

If the StarTeam Server you want to use does not appear on the list, right-click to add a new server or edit an existing server.

When you select a server, you're prompted to log on if you haven't already done so in this work session.

- 4 Log on to StarTeam, if necessary.

It's usual to use your network logon name. However, this name and password must be known to the server and you must have the necessary access rights to continue. See your StarTeam administrator for your server logon name.

- 5 To create a new StarTeam project to contain this JBuilder project, right-click the server node and choose New Project to create a new StarTeam project.

This opens the Create StarTeam Project dialog box, which lets you create a new project on the StarTeam Server and specify your default working folder.

- The StarTeam project name must be unique.
- The working folder stores locally the files that you check in and check out.

The directory specified in the Default Working Folder field is used as the default target directory when the project is pulled from StarTeam. Users can accept the default target directory or specify an alternate directory to use as the working folder when a project is pulled from StarTeam.

- 6 Look at the View Configuration option to see if it's what you want. Click the ellipsis (...) button to change it in the View Configuration dialog box.

StarTeam projects can have multiple views. The View Configuration dialog box lets you choose between the following:

- Current Configuration
- Labeled Configuration, with available labeled configurations available in a drop-down list
- Promotion State Configuration, with available promotion state configurations available in a drop-down list
- Configuration As Of a date and time you specify. Click the date to change it, and click the time fields to change them

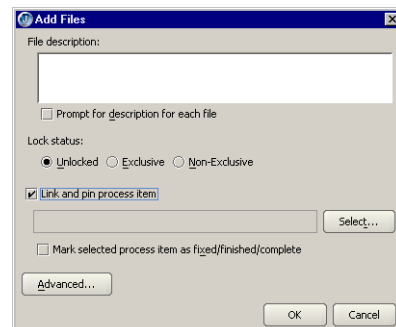
- 7 Click Next to go to the Select Directories And Files To Include/Exclude page.

- 8 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 include.

By default, the `bak` and `classes` directories are excluded and the `src` files are included. The JBuilder project (`.jpx`) file is required to be included because it manages team-wide version control settings and preferences as well as other project settings.

- 9 Click Finish.

The Add Files dialog box opens:



- 10 Optionally (but recommended), fill in the information in the Add Files dialog box:
  - a Type a generic description for all files in the File Description text box, or select the Prompt For Description For Each File check box to be prompted for separate descriptions for each file.
  - b Select an appropriate lock status for the files:
    - Unlocked: indicates you do not intend to make changes.
    - Exclusive: indicates you intend to make changes to these files, and prevents others from checking the files in. Unless another user breaks your lock, no one else can create a new revision of a locked file in the repository until you release your lock.
    - Non-Exclusive: indicates you are working on the files, and may possibly make changes. Other users can alter and check in the files.

Your lock choice lets other team members know whether or not you are working on the files. An exclusive lock means you intend to change the files.
  - c Check the Link And Pin Process Item check box if you want to link a process item to your files.
    - If process rules are enforced for this project, this option is required.
    - If the use of process items is required, the Link And Pin Process Item check box is selected by default.
    - If an active process item has been set, it's pre-selected as the process item to link.
    - To select a process item, click the Select button to open the Select Process Item dialog box.
    - If this process item is now fixed, finished, or complete as a result of placing the project into StarTeam, then check the Mark Selected Process Item As Fixed/Finished/Complete check box.
- 11 Optionally, specify advanced options:
  - a Click the Advanced button in the Add Files dialog box.

The Advanced Options dialog box opens.
  - b Determine how to handle the Delete Working Files option:
    - Check this check box to delete the files from your workstation, storing them only in the server configuration's repository
    - Uncheck this check box to keep the working files in the working folder as well as the repository.
  - c Uncheck Perform EOL Conversion if you do not want to convert each end-of-line (EOL) character for the files being added to StarTeam.

EOL conversion is selected by default. It converts the EOL characters to a carriage return and line feed combination, which is the EOL character used on the Windows platform.

- d Select a label from the Revision Label combo box or create a new revision label by typing its name.

Existing labels are listed in reverse chronological order based on the time they were created.

A label is useful if you plan to retrieve these files as a group later or if you will need this specific revision of any of the files.

- e Check the Override Current Storage Options check box if you want to change the storage options for the project's files. Storage options include compression type and whether or not to store revisions as deltas.

When Override Current Storage Options is selected, you can select the compression type for your files from the Compression drop-down list box:

- None, for no compression
- Maximize Speed, for the fastest possible compression of file revisions to improve server performance
- Default, a compromise between maximum compression and maximum speed
- Maximize Compression, for the largest compression of file revisions to save space on the server

By default, StarTeam stores only the changes that were made to files from one revision to another. When Override Current Storage Options is selected, you can uncheck the Store Revisions As Deltas check box to store the files in their entirety for each revision. Full revision storage is best for binary files and for certain text files (such as `.rtf` files) whose delta can be as large as the file itself.

- f If you want, specify the File Encoding which StarTeam should use to store characters. Available file encodings include flavors of Windows, UTF, ASCII, Unicode, MS, IBM, Mac, JIS, ISO, and many others.
- g Click OK to save your settings and close the Advanced Options dialog box, or click Cancel to close the dialog box without saving your changes.

- 12 Click OK in the Add Files dialog box to place the project into StarTeam.

The StarTeam message pane shows the progress of files being added. When all the files are added, a StarTeam Information dialog box indicates that the operation is complete, and a StarTeam Repository node appears in the project pane. The StarTeam Repository node provides access to the project view features of the StarTeam client.

## Pulling an existing project from StarTeam

---

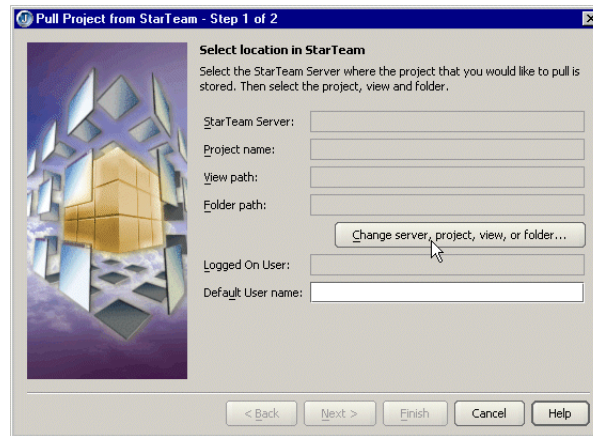
Pulling a project from a repository configures your connection to that project in the repository and deposits the project in your own workspace. In a team environment, it connects you to the network of users who can make changes in that project.

A project only needs to be pulled once. Once you have access to it, maintain it using the other commands available from the Team menu and from the context menus in the project pane and other areas of the development environment.

**Note** An instance of StarTeam Server (a server configuration) must be running before team members can use it. You can pull a project from any server configuration, so long as you have access to the server configuration and access rights to the project.

To pull a project from StarTeam,

- 1 Choose File|New|Project and double-click the Pull Project From StarTeam icon.  
The Pull Project from StarTeam wizard opens.
- 2 Click the Change Server, Project, View, Or Folder button.



The Select A Root Folder dialog box appears, showing a tree view which can be expanded to display available servers, projects, views, and folders.

- 3 Expand the nodes to navigate down to the project you want to open.  
If the StarTeam Server you want to use does not appear on the list, right-click to add a new server or edit an existing server.

When you select a server, you are prompted to log on.

- 4 Log on to StarTeam.

Usually people use their network logon name. However, this name and password must be known to the server and you must have the necessary access rights to continue. See your StarTeam administrator for your server log on name. After logging on, you can fill in the rest of the StarTeam project location information.

**Note** If the folder you specified does not contain a JBuilder project (.jpx) file, JBuilder creates a new project file when you complete the wizard. When new project files are created, they are not automatically added to StarTeam. You can add them later.

- 5 Click Next to select a target directory.
- 6 Type in a path to an empty local directory (new or existing) to store the project, or click the ellipsis (...) button to browse to a directory.

This directory will become the local workspace for the project. The default value in the Target Directory text box is based on the alternate working folder specified by the team member who placed the project into StarTeam.

**Tip** If you specify an invalid directory, a message appears in the wizard letting you know that there is a problem with that directory. In that case, type in a new directory, or click the ellipsis (...) button and try again.

- 7 Click Finish to complete the wizard.

The Pull Project From StarTeam Server dialog box appears and reports on the progress of the checkout.

- 8 Click Close to close the dialog box.

When JBuilder pulls a project from StarTeam, it adds a StarTeam Repository node to the root of the project in the project pane. This node lets you open the StarTeam project view window elements within JBuilder to provide extensive access to StarTeam features and information.

**Table 4.1** StarTeam: Project-wide version control status, files, and commands

Tool	Show status	Set files	Run commands
Project pane decorations	X		
Status Browser	X	X	
Commit Browser	X	X	X
File Include Lists		X	

The Status Browser is a viewing tool. The Commit Browser incorporates the viewing features of the Status Browser, and adds context-aware StarTeam command functionality.

Access the Status Browser or Commit Browser in one of these ways:

- Choose TeamIStatus Browser or TeamICommit Browser.
- Right-click the project node and choose either StarTeamIStatus Browser or StarTeamICommit Browser.
- Right-click the project directory in the Files tab of the project pane and choose either StarTeamIStatus Browser or StarTeamICommit Browser.

These browsers have three pages:

- Changes/Commits page
- StarTeam page
- File Include Lists page

When you're done with the Status Browser, click OK to close it. It's purely a viewing tool, so nothing is changed.

When you're done with the Commit Browser, click the Commit button at the bottom:

- All of the commands you specified for the displayed files get executed.
- All of your comments are applied as specified.
- Files with No Action specified are left unchanged in relation to StarTeam.
- The Commit Browser closes and a progress dialog appears, displaying the progress of the version control operations.

If updating a file from the Commit Browser creates a merge conflict, the conflicts are clearly marked with “<<<<<<”, “====”, and “>>>>>>”, and JBuilder’s merge conflict handling mechanisms are engaged.

## See also

- “Finding information and commands quickly” on page 13
- “Merge Conflicts page” in *Building Applications with JBuilder*

## Using the Changes and Commits pages

**In both Status  
Browser and Commit  
Browser**

These pages are very similar. The Status Browser uses the Changes page, which displays changes in files and directories. The Commit Browser uses the Commits page, which adds StarTeam commands and support for comments.

These pages consist of the following elements, starting in the upper left and going clockwise:

Element	Location	Status Browser	Commit Browser
Tree view	(Upper) left	X	X
File table	Upper right	X	X <sup>1</sup>
Summary Comment panel	Lower left		X
Tabbed source views	Lower right	X	X <sup>1</sup>

1. Indicates that there is added functionality for this element in the Commit Browser.

## Tree view

In both Status  
Browser and Commit  
Browser

The tree view displays all of the changed files in their hierarchy. In the Commit Browser, it also provides access to commands which you can apply recursively, against every file below the level of the selected node.

To display this	Do this
All the files in the project (shown in the file table)	Select the Full List node in the directory tree
Subdirectories in the directory tree	Expand the module or parent directories
Files for an individual directory (shown in the file table)	Select the directory node in the directory tree
<b>Additionally, in the Commit Browser,</b>	
Apply StarTeam commands recursively	Right-click a node

## Table of changed files

In both Status  
Browser and Commit  
Browser

This table lists the changed files which belong to the node selected in the tree view on the left side of this page. Select a file in this table to view its source in the tabs below it in the dialog box.

<b>Action</b> <sup>1</sup>	Provides context-sensitive StarTeam commands.
<b>Status</b>	Displays the file's current state in relation to StarTeam.
<b>File Name</b>	Displays the name of each file that has been changed in relation to StarTeam.

1. Only the Commit Browser uses the Action column. Both the Status Browser and the Commit Browser use the Status and File Name columns.

In the Commit Browser, the Action column provides StarTeam actions in a drop-down list for each file.

**Table 4.2** StarTeam Commit Browser: Version control options

Option	Description
Add	Add this file so it's stored in the repository.
Commit	Check in the change to the repository.
Delete	This file is not in the repository. Delete removes it from your workspace, so your workspace matches the repository.
Force Commit	This file has changed in the workspace, but is not based on the tip revision (most recent revision) in the repository. Unless there are merge conflicts, this option performs a forced check in, updating the repository with workspace version of the file. All the revisions of this file since you checked it out will be overwritten.
Force Update	This file has changed in the workspace, but is not based on the tip revision (most recent revision) in the repository. Unless there are merge conflicts, this option performs a forced check out, replacing the file in your workspace with the most recent revision in the repository.



**Table 4.2** StarTeam Commit Browser: Version control options (continued)

Option	Description
Get	This file has already been added to the repository; this checks it out to your workspace.
Handle Individually	<p>The StarTeam Server cannot determine the file's status. When you click Commit, the following error message appears in the StarTeam message pane:</p> <p>File "&lt;filename&gt;" has a status of unknown and update status failed to recognize the contents as a previous revision.</p> <p>You must use the force option on Checkin or Checkout to update the repository.</p>
Merge	This file has changed in the workspace, but is not based on the tip revision (most recent revision) in the repository. If there are no merge conflicts, the changes in the tip revision are merged into the file in the workspace when you click Commit. If there are merge conflicts, however, the merge operation will fail, and the user is prompted to use the file checkout operation to merge the contents manually.
No Action	This changed file will not be touched by a StarTeam operation of any kind. It will be exactly as you left it before you invoked the Commit Browser.
Rename Back To "<orig_filename>"	This file has been renamed or moved in the workspace. When you click Commit, the file in the workspace will be renamed back to its original name or returned to its original location, and no action is performed on the file in the repository.
Rename To "<new_filename>"	This file has been renamed or moved in the workspace. When you click Commit, the file in the repository will be renamed to the new name or moved to the new folder, and no action is performed on the file in the workspace.
Revert	Update the workspace with the latest repository version of this file, discarding all changes made since your last update.

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 Get.

Condition	Listed status	Default option	Alternative options
File changed in the workspace	Changed In Workspace	■ Commit	■ Revert ■ Exclude In Personal List ■ Exclude In Team List ■ No Action
File added to version control	Not In Repository	■ Add	■ Delete ■ Exclude In Personal List ■ Exclude In Team List ■ No Action
File deleted from version control	Not In Workspace	■ <i>If removed from within JBuilder:</i> Remove ■ <i>If removed from outside JBuilder:</i> Update	■ Exclude In Personal List ■ Exclude In Team List ■ Revert ■ No Action

**Note** When you add or remove files from the Commit Browser, they're automatically committed by JBuilder. When you add or remove files individually from a menu, they need to be committed in a separate step.

If you're solely interested in the Status Browser, skip ahead to [“Tabbed source views” on page 34](#).

## Summary Comment

**In the Commit Browser**

The Summary Comment panel is where you write a comment you want to apply to more than one file. To use it

- 1 Write a comment in this panel which applies to files in the file table.
- 2 As you go through the files in the table (upper right), check the Include Summary Comment check box at the bottom of the Individual Comment page (in the tabbed view) for the files you want to apply this summary comment to.
- 3 Optionally, write individual file comments under the Individual Comment tab.

Individual comments are appended to summary comments.

When you click the Commit button at the bottom of the page, JBuilder runs your StarTeam commands on those files and applies all comments as specified.

The Include Summary Comment check box at the bottom of the Individual Comment page allows you to specify whether or not to include the summary comment for individual files. This option is on by default, or when the individual comment is blank. If you uncheck Include Summary Comment for a file, only the individual comment will be applied when the file is committed.

When a summary comment is included, it appears before the individual comment, and both are maintained as a single comment for the revision. Summary comments are entered in the Summary Comment pane in the Commits page.

## Tabbed source views

**In both Status Browser and Commit Browser**

Select a file from the file table above to see its source and diffs displayed here. The appropriate source views for the file's StarTeam status become available. For instance, if the selected file was changed in the repository, the Workspace Source, Repository Source, and Repository Diff tabbed views become available.

**Table 4.3** StarTeam: Source views

Source view	Contents
Workspace Source	This file's source code from the current workspace version.
Repository Source	This file's source code from the current repository version.
Workspace Diff	This file's most recent changes in your workspace.
Repository Diff	This file's most recent changes in the repository.
Complete Diff	Differences between the current version of this file in the repository and the current version in your workspace.
<b>The Commit Browser adds one more tab:</b>	
Individual Comment	Enter your comments describing the version control action on the file selected in the file table above these tabs.

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 Include Summary Comment check box is used to determine whether the summary comment should be used, either as the sole comment or else prepended to the individual comment written for this file. This is described in [“Summary Comment” on page 34](#).

## Using the StarTeam page

### In the Commit Browser

The StarTeam page is similar to the StarTeam dialog boxes for checking in and adding files. This page is only available when the list of files contains files that can be added or checked in (committed).

1 Select an appropriate lock status for the file:

- Unlocked: indicates you do not intend to make changes.
- Exclusive: indicates you intend to make changes, and prevents others from checking the file in. Unless another user breaks your lock, he cannot create a new revision of a locked file in the repository until you release your lock.
- Non-Exclusive: indicates you are working on the file, and may possibly make changes. Other users can check in the file.
- Keep current: the checkin will keep each file's current lock status.

Your lock choice lets other team members know whether or not you are working on the file. An exclusive lock means you intend to change this file.

2 Check the Force Check-In option to check the file in even when it's older than the tip revision (or is otherwise unacceptable).

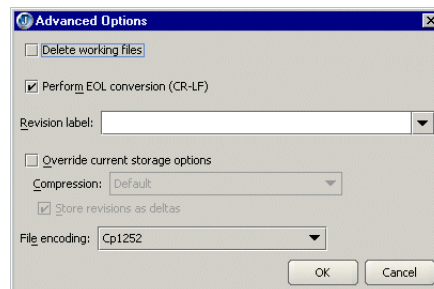
3 Check the Link And Pin Process Item check box if you want to link a process item to your files.

- If process rules are enforced for this project, this option is required.
- If the use of process items is required, the Link And Pin Process Item check box is selected by default.
- If an active process item has been set, it's pre-selected as the process item to link.
- To select a process item, click the Select button to open the Select Process Item dialog box.
- If this process item is now fixed, finished, or complete as a result of placing the project into StarTeam, then check the Mark Selected Process Item As Fixed/Finished/Complete check box.

4 Optionally, specify advanced options:

a Click the Advanced button.

The Advanced Options dialog box opens:



- b Determine how to handle the Delete Working Files option:
    - Check this check box to delete the files from your workstation, storing them only in the server configuration's repository
    - Uncheck this check box to keep the working files in the working folder as well as the repository.
  - c Uncheck Perform EOL Conversion if you do not want to convert each end-of-line (EOL) character for the files being added to StarTeam.

EOL conversion is selected by default. It converts the EOL characters to a carriage return and line feed combination, which is the EOL character used on the Windows platform.
  - d Select a label from the Revision Label combo box or create a new revision label by typing its name.

Existing labels are listed in reverse chronological order based on the time they were created.

A label is useful if you plan to retrieve these files as a group later or if you will need this specific revision of any of the files.
  - e Check the Override Current Storage Options check box if you want to change the storage options for the project's files. Storage options include compression type and whether or not to store revisions as deltas.

When Override Current Storage Options is selected, you can select the compression type for your files from the Compression drop-down list box:

    - None, for no compression
    - Maximize Speed, for the fastest possible compression of file revisions to improve server performance
    - Default, a compromise between maximum compression and maximum speed
    - Maximize Compression, for the largest compression of file revisions to save space on the server

By default, StarTeam stores only the changes that were made to files from one revision to another. When Override Current Storage Options is selected, you can uncheck the Store Revisions As Deltas check box to store the files in their entirety for each revision. Full revision storage is best for binary files and for certain text files (such as `.rtf` files) whose delta can be as large as the file itself.
  - f If you want, specify the File Encoding which StarTeam should use to store characters. Available file encodings include flavors of Windows, UTF, ASCII, Unicode, MS, IBM, Mac, JIS, ISO, and many others.
  - g Click OK to save your settings and close the Advanced Options dialog box, or click Cancel to close the dialog box without saving your changes.
- 5 If you're content with your settings in the Commits and File Include Lists pages, click the Commit button to perform all of the StarTeam operations specified in the Commit Browser.

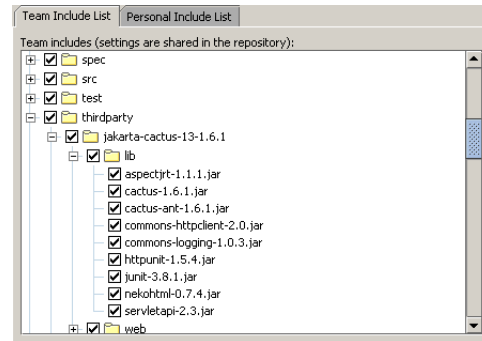
The StarTeam message pane shows the progress of the operations being performed. When all the files are added, a StarTeam Information dialog box indicates that the operations are complete.

## File Include Lists

**In both Status  
Browser and Commit  
Browser**

The File Include Lists determine which files are shared and which files are visible in your workspace. These are available from several places:

- TeamFile Include Lists
- File Include Lists page of the Status Browser
- File Include Lists page of the Commit Browser



There are two pages: Team Include List and Personal Include List.

- 1 Choose TeamFile Include Lists.
- 2 Use the Team Include List to determine which files are shared.
- 3 Use the Personal Include List to determine which files you want to keep in sight.

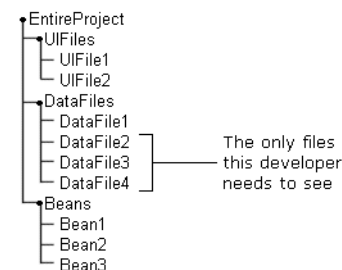
The files checked in the *Team Include List* are a team-wide project setting: the files that are checked here are the files that everyone needs to be able to use.

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 StarTeam in JBuilder.

The files checked in the *Personal Include List* are visible in your workspace; those that are unchecked are not, until you check them here. 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.

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



The Personal Include List page filters your view of the files, but does not affect the repository or the files themselves. Files that are checked on the Team Include List but not on the Personal Include List are still available on this page, and can be rechecked on the Personal Include List at any time.

## Synchronizing project settings

---

Each JBuilder project is administered by a project file. 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. This allows you to control when and whether global settings get posted to the database.

- 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.

**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. Child project files are updated with the rest of the project.

To maintain the project file in version control

- 1 Open the Sync Project Settings submenu, available from either of these places:
  - Team|Sync Project Settings
  - Right-click the project file in the project node and choose StarTeam|Sync Project Settings.
- 2 Pull or post the project file:
  - Sync Project Settings|Pull Latest "<projectfile>.jpx".
  - Sync Project Settings|Post Current "<projectfile>.jpx".

The dialog box for pulling or posting the project file appears.

- 3 Click OK to perform the operation on the project file.
- 4 Click OK to close the dialog box.

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.

### See also

- "Creating and managing projects" in *Building Applications with JBuilder* to understand project files better.

## Managing files in StarTeam

**StarTeam integration  
is a feature of  
JBuilder Developer  
and Enterprise**

The following sections describe how to use the StarTeam integration to manage files. Using the StarTeam integration, you can perform the following tasks:

- Add files to StarTeam
- Remove files from StarTeam
- Check in files
- Check out files
- Reconcile merge conflicts
- Compare file contents
- Lock and unlock files
- Locate files in StarTeam

**Note** The StarTeam Server must be running, your project must be under StarTeam control, and you must have appropriate access rights to perform the version control tasks described in the following sections.

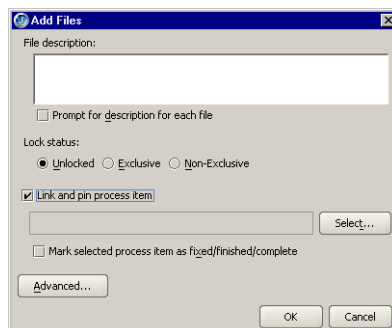
### Adding files

When you add a new file to your project, add it to StarTeam to put it under version control. The Add command adds files into StarTeam and logs the comment you type for the addition, as well as setting some version control properties, such as lock status and revision label.

To add the active file,

- 1 Choose Team|Add "<filename>".

The StarTeam Add Files dialog box appears:



2 Optionally (but recommended), fill in the information:

- a Type a description in the File Description text box.
- b If you're adding more than one file at a time, check or uncheck the Prompt For Description For Each File option.
- c Select an appropriate lock status for the file:
  - Unlocked: indicates you do not intend to make changes.
  - Exclusive: indicates you intend to make changes, and prevents others from checking the file in. Unless another user breaks your lock, he cannot create a new revision of a locked file in the repository until you release your lock.
  - Non-Exclusive: indicates you are working on the file, and may possibly make changes. Other users can check in the file.

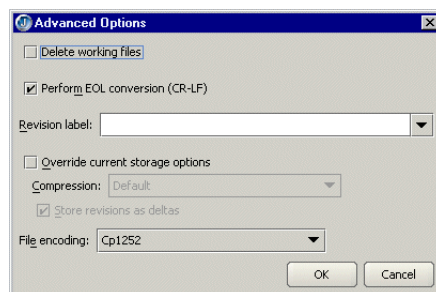
Your lock choice lets other team members know whether or not you are working on the file. An exclusive lock means you intend to change this file.

- d Check the Link And Pin Process Item check box if you want to link a process item to your files.
  - If process rules are enforced for this project, this option is required.
  - If the use of process items is required, the Link And Pin Process Item check box is selected by default.
  - If an active process item has been set, it's pre-selected as the process item to link.
  - To select a process item, click the Select button to open the Select Process Item dialog box.
  - If this process item is now fixed, finished, or complete as a result of placing the project into StarTeam, then check the Mark Selected Process Item As Fixed/Finished/Complete check box.

3 Optionally, specify advanced options:

- a Click the Advanced button.

The Advanced Options dialog box opens:



- b Determine how to handle the Delete Working Files option:

- Check this check box to delete the files from your workstation, storing them only in the server configuration's repository
- Uncheck this check box to keep the working files in the working folder as well as the repository.

- c Uncheck Perform EOL Conversion if you do not want to convert each end-of-line (EOL) character for the files being added to StarTeam.

EOL conversion is selected by default. It converts the EOL characters to a carriage return and line feed combination, which is the EOL character used on the Windows platform.



- d Select a label from the Revision Label combo box or create a new revision label by typing its name.

Existing labels are listed in reverse chronological order based on the time they were created.

A label is useful if you plan to retrieve these files as a group later or if you will need this specific revision of any of the files.

- e Check the Override Current Storage Options check box if you want to change the storage options for the project's files. Storage options include compression type and whether or not to store revisions as deltas.

When Override Current Storage Options is selected, you can select the compression type for your files from the Compression drop-down list box:

- None, for no compression
- Maximize Speed, for the fastest possible compression of file revisions to improve server performance
- Default, a compromise between maximum compression and maximum speed
- Maximize Compression, for the largest compression of file revisions to save space on the server

By default, StarTeam stores only the changes that were made to files from one revision to another. When Override Current Storage Options is selected, you can uncheck the Store Revisions As Deltas check box to store the files in their entirety for each revision. Full revision storage is best for binary files and for certain text files (such as .rtf files) whose delta can be as large as the file itself.

- f If you want, specify the File Encoding which StarTeam should use to store characters. Available file encodings include flavors of Windows, UTF, ASCII, Unicode, MS, IBM, Mac, JIS, ISO, and many others.
- g Click OK to save your settings and close the Advanced Options dialog box, or click Cancel to close the dialog box without saving your changes.

- 4 Click OK in the Add Files dialog box to add files into StarTeam.

The StarTeam message pane shows the progress of files being added. When all the files are added, a StarTeam Information dialog box indicates that the operation is complete.

## Removing files

---

If you remove a file from your project, you should also remove it from StarTeam. The Remove command removes the file from StarTeam control.

To remove the active file,

- 1 Choose Team | Remove "<filename>".

A Remove Files dialog box appears, notifying you that this will remove the file from your project and from StarTeam and will delete it from your workspace.

- 2 Click Yes to remove the file.

The file is removed from your project, deleted from your workspace, and is no longer under StarTeam version control.

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 StarTeam\Remove from the context menu.  
A Remove Files dialog box appears.
- 3 Click Yes to remove the files.  
The files are removed from your project, deleted from your workspace, and are no longer under StarTeam version control.

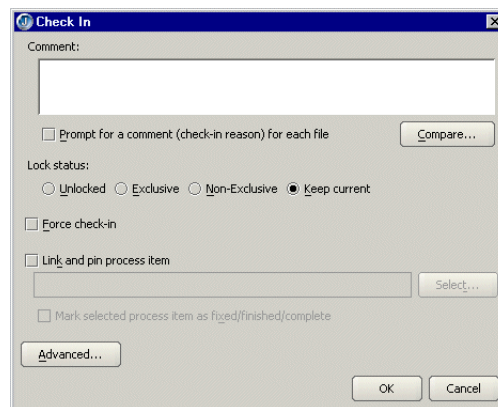
## Checking in files

Checking in files updates the repository with changes you made to the checked-out file or files.

To check in the active file,

- 1 Choose Team\Check In "<filename>".

The StarTeam Check In dialog box appears:

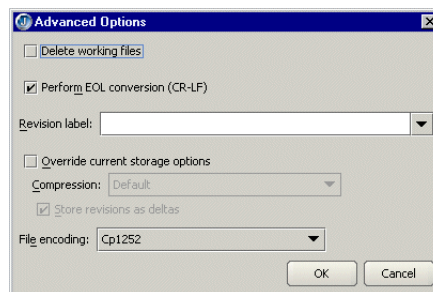


- 2 Optionally (but recommended), fill in the information:
  - a Type a description of the change in the Comment text box.
  - b If you're checking in more than one file and you want to comment on each one separately, check Prompt For A Comment (Check-In Reason) For Each File.
  - c Click the Compare button to bring up StarTeam's Visual Diff dialog box so you can see the changes you made since checking this file out.
  - d Select an appropriate lock status for the file:
    - Unlocked: indicates you do not intend to make changes.
    - Exclusive: indicates you intend to make changes, and prevents others from checking the file in. Unless another user breaks your lock, he cannot create a new revision of a locked file in the repository until you release your lock.
    - Non-Exclusive: indicates you are working on the file, and may possibly make changes. Other users can check in the file.
    - Keep current: the checkin will keep each file's current lock status.

Your lock choice lets other team members know whether or not you are working on the file. An exclusive lock means you intend to change this file.
  - e Check the Force Check-In option to check the file in even when it's older than the tip revision (or is otherwise unacceptable).

- f Check the Link And Pin Process Item check box if you want to link a process item to your files.
  - If process rules are enforced for this project, this option is required.
  - If the use of process items is required, the Link And Pin Process Item check box is selected by default.
  - If an active process item has been set, it's pre-selected as the process item to link.
  - To select a process item, click the Select button to open the Select Process Item dialog box.
  - If this process item is now fixed, finished, or complete as a result of placing the project into StarTeam, then check the Mark Selected Process Item As Fixed/Finished/Complete check box.
- 3 Optionally, specify advanced options:
  - a Click the Advanced button.

The Advanced Options dialog box opens:



- b Determine how to handle the Delete Working Files option:
  - Check this check box to delete the files from your workstation, storing them only in the server configuration's repository
  - Uncheck this check box to keep the working files in the working folder as well as the repository.
- c Uncheck Perform EOL Conversion if you do not want to convert each end-of-line (EOL) character for the files being added to StarTeam.
 

EOL conversion is selected by default. It converts the EOL characters to a carriage return and line feed combination, which is the EOL character used on the Windows platform.
- d Select a label from the Revision Label combo box or create a new revision label by typing its name.
 

Existing labels are listed in reverse chronological order based on the time they were created.

A label is useful if you plan to retrieve these files as a group later or if you will need this specific revision of any of the files.
- e Check the Override Current Storage Options check box if you want to change the storage options for the project's files. Storage options include compression type and whether or not to store revisions as deltas.
 

When Override Current Storage Options is selected, you can select the compression type for your files from the Compression drop-down list box:

  - None, for no compression
  - Maximize Speed, for the fastest possible compression of file revisions to improve server performance

- Default, a compromise between maximum compression and maximum speed
- Maximize Compression, for the largest compression of file revisions to save space on the server

By default, StarTeam stores only the changes that were made to files from one revision to another. When Override Current Storage Options is selected, you can uncheck the Store Revisions As Deltas check box to store the files in their entirety for each revision. Full revision storage is best for binary files and for certain text files (such as .rtf files) whose delta can be as large as the file itself.

- f If you want, specify the File Encoding which StarTeam should use to store characters. Available file encodings include flavors of Windows, UTF, ASCII, Unicode, MS, IBM, Mac, JIS, ISO, and many others.
  - g Click OK to save your settings and close the Advanced Options dialog box, or click Cancel to close the dialog box without saving your changes.
- 4 Click OK in the Check In dialog box to check the file changes into StarTeam.

When the operation is complete, the StarTeam message pane shows that the file has been checked in.

## Checking out files

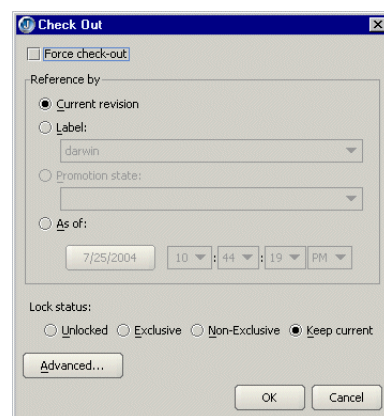
When you check out a file, StarTeam copies the requested revision of that file to the appropriate working folder. If a copy of that file is already in the working folder, it is overwritten unless the working file appears to be more recent than the checked in revision. In that case, you are asked to confirm the check out.

To check out the active file,

- 1 Choose Team|Check Out "<filename>".

The StarTeam Check Out dialog box appears.

**Tip** If you are only interested in checking out the tip revision of the file, click OK now, and you are done.



- 2 Optionally (but recommended), specify the checkout conditions:
  - a Select Force Check-out to overwrite any working files, even if they are less recent than those in the working folder.
  - b Select one of the following options in the Reference By group box:
    - Current Revision: to check out the tip revision of each selected file.
    - Label: to check out the revision of each of the selected files that has a specific view or revision label. (If the selected file does not have the label, no revision is checked out for that file.) Select the label from the Label drop-down list box.

**Note**

The existing view and revision labels are listed in reverse chronological order based on the time for which they were created. The view labels precede the revision labels in the list.

- **Promotion State:** to check out the revision of each selected file that has a specific promotion state. Actually, these are the revisions that have the view label currently assigned to this promotion state.
- **As Of:** to check out the revision that was the tip revision at the specified date and time.

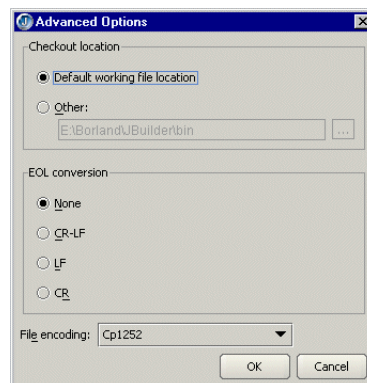
Click the date to use the calendar. Specify the date using the next and previous month buttons and the arrow keys. To specify the time, type the time or use the spin boxes.

- c Select one of the following options to specify the locking status for the checked out file:
- **Unlocked:** to release your lock on this file after checkout.
  - **Exclusive:** to indicate that you intend to make further changes to this file and prevent others from checking in this file.
  - **Non-Exclusive:** to indicate that you are working on the file and may possibly change it.
  - **Keep current:** to indicate that the checkout will keep the file's current lock status.

3 Optionally, specify advanced options:

- a Click the Advanced button in the Check Out dialog box.

The Advanced Options dialog box appears.



- b To check the file out to a folder other than the default working folder, click the Other radio button, and type or browse for a new folder for the checked-out file.
- c Select the type of end-of-line (EOL) conversion to perform when the file is checked out:
- **None:** no EOL conversion
  - **CR-LF:** carriage return/line feed (Windows)
  - **LF:** line feed (UNIX)
  - **CR:** carriage return (some other operating systems)
- d If you want, specify the File Encoding which StarTeam should use to store characters. Available file encodings include flavors of Windows, UTF, ASCII, Unicode, MS, IBM, Mac, JIS, ISO, and many others.
- e Click OK to save your settings and close the Advanced Options dialog box, or click Cancel to close the dialog box without saving your changes.
- 4 Click OK in the Check Out dialog box to check out the file.

When the operation is complete, the StarTeam message pane shows that the file has been checked out.

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 StarTeam|Check Out from the context menu.  
The StarTeam Check Out dialog box appears.
- 3 Optionally (but recommended), specify the checkout conditions as described above.
- 4 Click OK in the Check Out dialog box to check out the files.

When the operation is complete, the StarTeam message pane shows that the files have been checked out.

## Moving files

---

Moving files can be tricky in version control because you may lose all of the file's history up to the point of the move; as StarTeam would say, the file is not in view.

This StarTeam integration takes advantage of a useful form of push technology that makes your workspace aware of server changes before you ask for them. This allows StarTeam to check a status record of when the file *was* in view, and also check the status of that file (before and after the move) in your workspace and on the server. This second check is to preserve all changes and merge the workspace and server versions if needed.

To move files in StarTeam,

- 1 Move a file in JBuilder, for instance by dragging and dropping it in the project pane.
- 2 Check it in as you normally would.  
JBuilder displays a dialog asking you to confirm the refactorings necessary to accomodate the move.
- 3 Click OK in the confirmation dialog box.  
A Check In dialog box appears.
- 4 Fill in the fields in the Check In dialog as you normally would.
- 5 Click OK when you're done.
- 6 Click OK in the confirmation dialog box to return to work in the IDE.

## Merge conflicts

---

JBuilder helps avoid merge conflicts by requiring you to update when necessary before checking in changes. If merge conflicts do occur when you attempt to merge a file, StarTeam and JBuilder alert you to the conflict, and provide a means to reconcile the merge conflicts. The merge conflict handling is slightly different when you are using the Merge action in the Commit Browser versus the Check Out command on the Team menu.

If you use the Commit Browser to perform a merge, and merge conflicts occur,

- The Version Control Operation dialog box reports that the file has been merged in the workspace, and that it contains conflicts.
- StarTeam inserts “<<<<<<”, “=====”, and “>>>>>>” around the conflicting blocks in the file source in JBuilder's editor. 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:

```
<<<<<< LocalFile
// Comment modified in file in the workspace
=====
// Comment in file checked into repository
>>>>>> ServerFile
```

- The message pane opens and lists the files that contain merge conflicts. When you select a listed file in the message pane, the conflicts are displayed in the Merge Conflicts page of the history view.
- JBuilder's Merge Conflicts page in the history view displays the workspace source (local file) side-by-side with the repository source (server file), with the conflicting blocks of code or text highlighted. Buttons 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.

If you use the check out command on the Team menu to perform a merge, and merge conflicts occur,

- StarTeam displays a warning, indicating that merge conflicts have been found. Click OK to close the warning.
- StarTeam opens a text editing window containing the file source. StarTeam inserts "<<<<<<", "=====", and ">>>>>>" around the conflicting blocks in the file source. 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:

```
<<<<<< C:\jbproject\Framel.java (local)
// Comment modified in file in the workspace
=====
// Comment in file checked into repository
>>>>>> C:\jbproject\Framel.java, Branch Revision: 1.3
```

## Reconciling merge conflicts

---

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

- 1 Reconcile them manually, searching for the "<<<<<<", "=====", and ">>>>>>" tagging in the file source, and using the text editor supplied by StarTeam or the editor in JBuilder to resolve the conflicts. Remember to delete the conflict tagging and any extraneous text when you're done.
- 2 Reconcile them automatically, using the buttons in the Merge Conflicts page in the history view to choose which versions of the conflicting areas to keep. The Merge Conflicts page is enabled only when JBuilder detects a conflict.

Choose the way that best addresses the conflict.

### Reconciling conflicts manually

When StarTeam finds merge conflicts when you use the check out command (Team! Check Out "<filename>"), it automatically opens a text editing window containing the file source. The conflicting lines of code are tagged in the file source displayed in the window.

You can edit the source code in this window and save your changes to reconcile conflicts, or you can close the window (saving the file) and reconcile the conflicts using JBuilder's editor or JBuilder's Merge Conflicts page. If you close the text editing window without saving the file, the file is returned to the state it was in prior to the attempted merge.

When merge conflicts are found when you use the Commit Browser, the file source is merged in the workspace and the merged source is displayed in the editor. The conflicting lines of code are tagged in the file source. You can reconcile the conflicts manually using JBuilder's editor, or automatically using JBuilder's Merge Conflicts page.

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.*;
<<<<<<< C:\jbproject\Framel.java
// fifth change
=====  
// conflicts with the fifth change
>>>>>>> C:\jbproject\Framel.java, Branch Revision: 1.3
/**
 * <p>Title: </p>
 * <p>Description: </p>
```

- 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.*;
// fifth change
/**
 * <p>Title: </p>
 * <p>Description: </p>
```

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

- 3 Save the file and check in the changes.

In the text editing window, save the file by exiting (File|Exit) and clicking Yes in the dialog box that appears.

**Note** Once conflicts have been tagged, StarTeam 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.

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

## Comparing file revisions

---

There are several ways of comparing file revisions in the StarTeam integration for JBuilder:

- Team|Diff and Team|Workspace Diff support the most common file comparison use cases.
- Using the upper and lower panes of the project view window, you can make more distinct comparisons.

To compare the contents of the active working file with the contents of the latest (tip) revision in the repository, choose Team|Diff "<filename>". This opens StarTeam's



Visual Diff tool, displaying the tip revision of the file on the left and the working version on the right. Differences between the files appear in a different color.

To compare the contents of the active working file with the contents of the revision that was checked out, choose Team|Workspace Diff "<filename>". This opens the Visual Diff tool, displaying the checked-out revision of the file on the left and the working version on the right.

You can also use JBuilder's history view, which displays diffs, comments, line-by-line annotations, and complete history information on version control and local backup versions.

Using the context menus in the project view window, you can compare the contents of any two files. You can also compare the properties of items. Refer to the *StarTeam User's Guide* for information on using Visual Diff and the context menus to compare files and file revisions.

### See also

- "Comparing files and versions" in *Building Applications with JBuilder* for information on using JBuilder's tools for comparing file revisions
- "Comparing Two Revisions of the Same File" in *StarTeam User's Guide*

## Locking and unlocking files in StarTeam

---

File locking is a way to inform other developers that you are revising a file (exclusive lock) or thinking about revising it (non-exclusive lock). File locking can be specified when files are checked in and out, or when they are added.

To lock or unlock the active file,

- 1 Choose Team|Lock/Unlock "<filename>".

The Set My Lock Status dialog box appears:



- 2 Select a lock status option:

- Unlocked to remove your exclusive or non-exclusive lock on the selected items
- Exclusive so that no one can create a new revision of this item except you (until you release the lock or someone breaks your lock)
- Non-exclusive to indicate that you are working on the item and may possibly make changes to the item (non-exclusive locks are not recommended for items other than files)

Depending on your privileges regarding a selected item, you may be able to break another team member's lock on that item.

- 3 To break a lock, check the Break Existing Lock check box.
- 4 Click OK to close the dialog box.

You can also set the lock status at any time with the Set My Lock Status dialog box.


**Note** Depending on your personal options (Team|Personal Options), you may have unlocked files that are marked read-only. This prevents you from inadvertently making changes to files that you have not locked. Read-only files are indicated in the project pane by read-only decorations added to the file icon.

### See also

- "Locking and Unlocking Items" in *StarTeam User's Guide*

## Finding files in StarTeam

---

The quickest way to locate files in StarTeam is to select the file in the project pane, and click the Find button  on the toolbar of the StarTeam Repository pane. This locates and selects the file in the project view window's upper pane.

## Working with process items and topics in StarTeam

**StarTeam integration is a feature of JBuilder Developer and Enterprise**

Whether or not process rules are required, you can take advantage of the linking and tracking made possible with process items such as change requests, requirements, and tasks. As you add files or check them in, you can indicate that the new file revisions are to be linked and pinned to a specific process item. Do this by selecting a change request, requirement, or a task as the process item for the operation. At the same time, you can mark the change request as fixed, the requirement as complete, or the task as finished.

Using process items allows you to clearly distinguish the following:

- Which file revisions are related to or fix a specific change request
- Which file revisions are related to or complete a specific requirement
- Which file revisions are related to or finish a specific task

Topics provide threaded conversations that you can place in specific project folders and link to specific project items, such as files, folders, or process items. For example, you can link a topic to the change request created as a result of the discussion as well as to the document or file that was changed to comply with that change request.

The following sections describe how to perform basic process item operations in the StarTeam integration. Using the StarTeam integration, you can perform the following operations:

- Add, remove, and link process items and topics
- Edit process items and topics
- Create shortcuts
- Set the active process item

**Important**

The StarTeam project view window is incorporated into JBuilder as part of the StarTeam integration. When you open the StarTeam Repository node, the upper and lower panes of the project view window appear in the StarTeam Repository pane, and the folder hierarchy pane opens to the left of the StarTeam Repository pane. These panes and the context menus in these panes provide comprehensive access to commands and information associated with process items. However, you must use the StarTeam client (TeamLaunch StarTeam) to specify process rules or enable the use of alternate property editors (APEs). Please refer to the *StarTeam User's Guide* for detailed information on how to use process items and process rules.

**Note** The StarTeam Server must be running, your project must be under StarTeam control, and you must have appropriate access rights to perform the version control tasks described in the following sections.

## Adding, linking, and removing process items

---

In the StarTeam integration for JBuilder, you work with process items and topics, just as you would in the StarTeam client. Process items can be added, linked, and removed, using commands on the context menu for the appropriate page of the upper pane of the project view window.

Please refer to the *StarTeam User's Guide* for detailed instruction and usage information for working with process items and topics.

### See also

- “Using process items” in *StarTeam User's Guide*
- “Understanding topics” in *StarTeam User's Guide*

## Editing process items

---

You can edit process items and topics within JBuilder either the same way you would edit them from within the StarTeam client or by using the item's shortcut in the project pane:

- Double-click the shortcut.
- Right-click it and choose Open from the context menu.

Any of these methods opens a multi-page property editor for the process item in the content pane. These pages function as an embedded dialog box, allowing you to work on your source files and edit process item properties simultaneously.

**Tip** When a change request is read-only, its change-request icon in the project pane has a read-only figure added to it.

**Note** Depending on how your team has set up StarTeam, you may see a different dialog box called an alternate property editor (APE). APEs are created with StarTeam Extensions. Refer to the *StarTeam Extensions User's Guide* for more information about APEs and workflow processes.

### See also

- “Using process items” in *StarTeam User's Guide*

## Setting the active process item

---

Selecting an active process item is a convenience that can save you time as you add files or check them in later. The active process item becomes the default selection for a process item in the Add Files and Check In dialog boxes. Within JBuilder, there are two ways to set a process item as the active process item, using the StarTeam integration: using the context menu in the upper pane of the project view window, or using the context menu for shortcut nodes.

To set the active process item in the upper pane of the project view window,


- 1 Open the StarTeam Repository node in the project pane.

This opens the StarTeam project view window elements in the JBuilder user interface. The upper and lower panes of the project view window appear in the StarTeam Repository pane. The folder hierarchy appears below the project pane.

- 2 In the upper pane of the project view window, select the process item (change request, requirement, or task) you want to set as the active process item.
- 3 Right-click the process item, and choose Set Active Process Item from the context menu.

To set the active process item using a shortcut,

- 1 Select the shortcut for the process item you want to set as the active process item.
- 2 Right-click the shortcut, and choose Set Active Process Item from the context menu.

The active process item icon (  ) appears in the shortcut.

**Note** Selecting a second active process item clears the first. There is also a Clear Active Process Item command on the Change Request, Requirement, and Task menus, but you will probably never use it. You do not have to use the active process item while adding files or checking them in. The active process item becomes the default selection for a process item, but you can select another appropriate item.



# Performing StarTeam administrative tasks

**StarTeam integration  
is a feature of  
JBuilder Developer  
and Enterprise**

Most administrative tasks, such as creating more project views, managing accounts and users, or adding and configuring servers, should be handled through the StarTeam client and native StarTeam tools. The following sections describe how to use some of the administrative features that are available from within JBuilder.

## Personal Options

---

The StarTeam Personal Options dialog box contains the following pages:

- **Workspace:** lets you specify confirmation requirements for version control operations, display options, and other parameters that apply to the behavior and appearance of StarTeam item in the workspace.
- **StarTeamMPX Server:** lets you enable support for StarTeamMPX Server for the active project and subsequently opened projects. When StarTeamMPX Server is enabled, information about changed objects in a StarTeam server configuration is broadcast in encrypted format through a publish/subscribe channel to the StarTeam client. The caching modules in the StarTeam client automatically display the new information to the user. By reducing demand on the StarTeam Server, the caching modules also increase the number of active sessions that can be effectively managed in a single server configuration.
- **File:** lets you set checkout options, locking options, merging options, end-of-line options, default file status repository settings, and alternate applications for editing, merging, or comparing files.
- **Change Request:** lets you set system tray notification parameters and locking options for change requests.
- **Requirement:** lets you set system tray notification parameters and locking options for requirements.
- **Task:** lets you set system tray notification parameters and locking options for tasks.
- **Topic:** lets you set system tray notification parameters and locking options for topics.

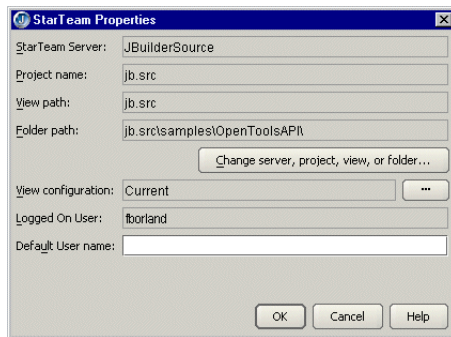
The StarTeam Personal Options dialog box can also be opened in the StarTeam client. Please refer to the *StarTeam User's Guide* for additional information on setting personal options.

**Note** StarTeamMPX Server is a part of StarTeam Enterprise Advantage, but it can be purchased separately with StarTeam Standard and StarTeam Enterprise. For more information, refer to the *StarTeamMPX Server Administrator's Guide*.

## Viewing StarTeam connection properties

---

The connection to the version control system for a JBuilder project 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 place a project into the repository for the first time from within JBuilder. After the connection has been set, you can view and change it from this dialog box.



**Note** The name of a StarTeam project, its root view, and the root view's root folder are often identical.

## Logging on and off

---

Users can log off of the StarTeam Server in two ways. When users close JBuilder the Integration automatically logs them off of the StarTeam Server. Or, choose TeamLog "<user>" Off StarTeam Server to explicitly log off the server.

When you are logged off, and you attempt to perform a source control operation from JBuilder, the Integration prompts you to log on. Alternatively, you can choose TeamLog "<user>" On To StarTeam Server to log back on explicitly.

**Note** If you have a floating license for StarTeam, sometimes called a concurrent license, there is usually a limit to the number of concurrent connections allowed to the StarTeam Server. You can log off (TeamLog "<user>" Off StarTeam Server) to free a connection for another team member. If you have opened the StarTeam Cross-Platform Client (TeamLaunch StarTeam), you must also close the client in addition to logging off.

## Creating and changing views

---


To create a new view:

- 1 Choose TeamConfigure StarTeam.
- 2 Click the button that says Change Server, Project, View, Or Folder.

The Select A Root Folder dialog box appears.



- 3 Right-click a view node and choose New View.

View nodes are distinguished by the View icon: 

- 4 Specify its properties in the New View wizard.
- 5 Click Finish when you're done with the wizard.

You may need to have appropriate access rights to create a new view.

You can also roll back a view to a past state based on a label, promotion state, or a point in time. However, this “freezes” the view until you change its configuration back to current or close the project (which automatically changes the configuration back to current). You are not allowed to check in files, update change requests, and so on because this would effectively change the past.

### See also

- “Understanding Views” in *StarTeam User's Guide* for more on creating views and for forward links to different uses of the New View wizard.
- “Configuring a View” in *StarTeam User's Guide* for how to roll back a view and how to return to the current configuration.


## Using labels

---

There are two types of labels used in StarTeam:

View labels	Generally speaking, these “time stamp” the entire contents of the view. When you roll back the view to that label, you see everything that existed at that point in time, unless the label has been adjusted.
Revision labels	These are not attached automatically to any item in the view. The purpose of a revision label is for use with a set of folders or items within the view. For example, you might want to label a group of files that should be checked in and out together.

To create a new view label:

- 1 Click the Labels button  in the StarTeam Repository pane to open the Labels dialog box.

The Labels dialog box lists existing labels and their descriptions in reverse chronological order, based on the time at which they were created.

- 2 Select the View tab at the top of the dialog box.
- 3 Click the New button.

The View Label dialog box opens.

- 4 Fill in the label information in the View Label dialog box:

- a Type a name and description for the label in the appropriate text boxes.

The maximum label name length is 64 characters and the description length is 254 characters.

- b Select one of the following option buttons:

- Current Configuration to attach the label to the tip revision of every item in this view's current configuration.
- Labeled Configuration to attach the label to the revision with the label you specify. The existing view labels are listed in reverse chronological order based on the time for which they were created.

- Promotion State Configuration to attach the label to the revision currently in the promotion state that you specify (actually, the label is attached to the revision that also has the promotion state's current view label).
- Configuration As Of to attach the label to the revision that was the tip revision at the specified date and time.
- c Optionally, select the Use As Build Label check box to update each change request that has "Next Build" as the setting for its Addressed In Build property.  
If this option is not selected, change requests will still have this label, but it will not affect the setting of the Addressed In Build property. Only view labels that are designated for use as build labels appear in the Last Build Tested and Addressed In Build drop-down lists for change requests.
- d Optionally, to freeze this label so that the revisions attached to it cannot be changed, check the Frozen check box.
- e Click OK to close the View Label dialog box and create the view label.

**Note** The computers that run StarTeam Server and clients must have their dates and times synchronized. Many features of StarTeam depend on calculations involving times and dates; in particular, labels, configurations and promotion states are all governed by time and date calculations. If the client and server are not kept synchronized, a number of operations may fail, or produce inaccurate or unreliable operations such as checkout, file status displays or label creation.

To create a new revision label:

- 1 Click the Labels button in the StarTeam Repository pane to open the View Configuration dialog box.  
The Labels dialog box lists the existing labels (along with their descriptions) in reverse chronological order based on the time at which they were created.
- 2 Select the Revision tab and click the New button.  
The Revision Label dialog box opens.
- 3 Type a name and description for the label in the appropriate text boxes.  
The maximum name length is 64 characters and description length is 254 characters.
- 4 If you want to base this label on a label that already exists,
  - a Check the Copy From Another Revision Label check box.
  - b Click Select to open the Copy A Revision Label dialog box.
  - c Select the project, view, and label you want to copy.
  - d Click OK to close this dialog box.
- 5 Optionally, to freeze this label so that the revisions attached to it cannot be changed, select the Frozen check box.
- 6 Click OK to exit the Revision Label dialog box.

The Labels dialog box also lets you delete, freeze, and unfreeze labels, and view and change their properties.

There are numerous tasks associated with labels. Please refer to the *StarTeam User's Guide* for more information on using labels.

**See also, in *StarTeam User's Guide***

- "Understanding Labels"
- "Attaching Existing Labels"
- "Creating View Labels"
- "Creating Revision Labels"
- "Freezing or Unfreezing Labels"

## Managing shortcuts

You can create shortcuts for folders, change requests, and other process items. These shortcuts appear in the project pane as children under the StarTeam Repository node.

To add a shortcut for a folder or item,

- 1 Open the StarTeam Repository node in the project pane.

This opens the StarTeam project view window elements in the JBuilder user interface.






- 2 In the upper pane of the project view window, select the item for which you want to create a shortcut.

- 3 Click the Item Shortcut button in the toolbar at the top of the StarTeam Repository pane.

The item's shortcut automatically appears as a subnode of the StarTeam Repository node in the project pane.

**Tip** To create a shortcut for a file, use a folder shortcut.

The shortcuts for different types of elements are identified in the project pane by their icons: each one consists of its node icon with the shortcut icon attached.

Icon	Shortcut type
	Folder Shortcut
	Item: Change Request
	Item: Requirement
	Item: Task
	Item: Topic


Organize, rename, and delete shortcuts from these context menus:

- In the folder tree on the Shortcut tab
- In the shortcut's node in the project pane

**Note** Shortcuts apply to the local workspace only, and are not shared between team members.

## Creating process item shortcuts

Specifically, shortcuts for process items and topics provide quick and comprehensive access to property editors for process items and topics. Shortcuts let you open property editors for items in the content pane, instead of the default modal dialog box.

A gears icon (  ) in the shortcut node indicates the active process item. The active process item (API) appears as a shortcut as soon as the API is set, but changes back to its original appearance in the project pane unless you made it a shortcut before setting it as the API.

The context menus for process item shortcuts add Set As Process Item/Clear As Process Item to the commands described above.



# Version control reference: StarTeam

**StarTeam integration  
is a feature of  
JBuilder Developer  
and Enterprise**

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

## Version control system API

---

The JBuilder IDE includes a framework for integrating additional version control systems (VCSs), and using the VCS from within the IDE to manage Java projects. This capability is described by the VCS API, which you can use to customize an existing integration or to design your own.

We recommend you start by reading “Introduction to PrimeTime and JBuilder OpenTools” and “OpenTools basics” in *Developing OpenTools* to familiarize yourself with the structure and use of the OpenTools API documentation.

To learn how to integrate your own version control tool into the JBuilder IDE, or customize the existing integrations, refer to “Version control system concepts” in *Developing OpenTools* and the `com.borland.primetime.teamdev.frontend` and `com.borland.primetime.teamdev.vcs` packages.

## 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).

## StarTeam resources

---

The following resource provides additional information about StarTeam.

- Home page: <http://www.borland.com/starteam/index.html>



Part II

# Using CVS with JBuilder

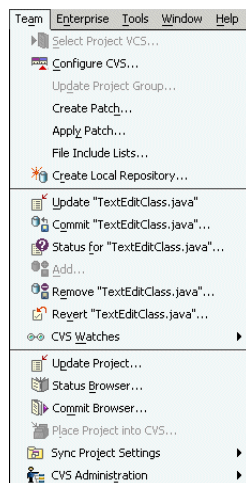




## CVS in JBuilder

Concurrent Versions System (CVS), a widely used open source version control system, is integrated with all editions of JBuilder. When you install JBuilder, CVS is automatically installed into the `<jbuilder>/bin` directory. JBuilder has a responsive interface that provides context-sensitive access to the most common CVS commands from within the AppBrowser.

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 66.)



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 the [“Handling binary files in CVS”](#) on page 97.

**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.

## Finding information and commands quickly

---

JBuilder provides access to relevant version control information and commands from several convenient places, supporting many different working styles.

The history view in the content pane lets you examine prior revisions, look at diffs, read revision lists and comments, view checkins line by line, and handle merge conflicts if they arise.

In the project pane and the file tabs, JBuilder decorates the icons of files changed in version control. It provides textual notations and tool tips in the project pane describing each changed file's state. It checks the repository state every 15 minutes by default. Customize these settings in the Tools|Preferences dialog box, on the Project tab's Decorations page.

Pertinent version control commands are available in these places:

- Choose the Team menu on the main menu bar for a complete list of commands.
- In the project pane, right-click these nodes:
  - Project node, in the Project tab
  - File node, in the Project tab
  - Any directory that's part of the project, in the File tab
- To pull a project from the repository, choose File|New|Project.

## CVS glossary

---

These words have specific meanings in CVS:

Add	Schedules a file to be added to the module. JBuilder automatically follows this with a Commit, so the file goes straight into the repository when added in this IDE.
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.



# Chapter 10

## Working on an existing project in CVS

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.

**Tip** In the project pane, JBuilder decorates the icons of files changed in version control. Customize these settings in Tools|Preferences, Project tab, Decorations page.

**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

To check out a project that's already in the repository, use the Pull Project From CVS wizard available in the object gallery:

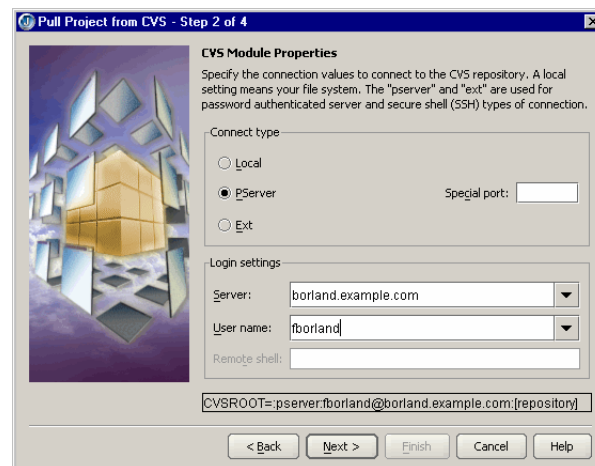
- 1 Choose File|New|Project to open the Project panel in the object gallery.
- 2 Select Pull Project From CVS .
- 3 Either click OK, double-click the Pull Project From CVS icon, or press *Enter* to launch the Pull Project From CVS wizard.
- 4 Type or select an empty target directory.

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 Select the type of checkout operation you want to perform, Checkout or Export.

If you intend to do development on the files, and want the files to be under CVS version control, select Checkout. If you intend to build a software release, or if for any other reason you want to check the files out without CVS administrative files or directories, select Export.

- 6 Click Next to proceed to Step 2.



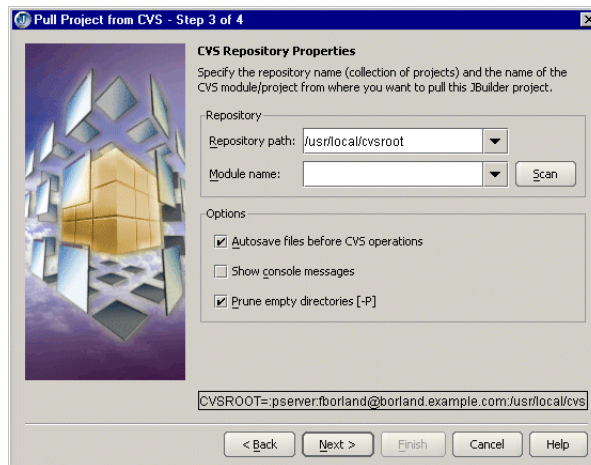
Under Connection Type, *Local* connects to a local repository, *PServer* connects to a repository on a server that's normally password-protected, and *Ext* connects to a repository on a secure server. If you select *PServer*, you have the option of specifying a port number for connecting to the remote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

**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.

## 7 Make your entries and click Next to proceed to Step 3.



## 8 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.

## 9 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.

## 10 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.

## 11 Optionally, you can check Autosave Files Before CVS Operations, Show Console Messages, or Prune Empty Directories.

- 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.
- 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.
- Prune Empty Directories works the same as the `-P` CVS command-line option. If a directory is empty, it will not be pulled to your local workspace.

## 12 Click Next to go to the Select CVS Branch Or Date step and make your selections.

This step of the wizard lets you specify a branch, a version label, or a specific date when you pull files.

The following four options can be selected:

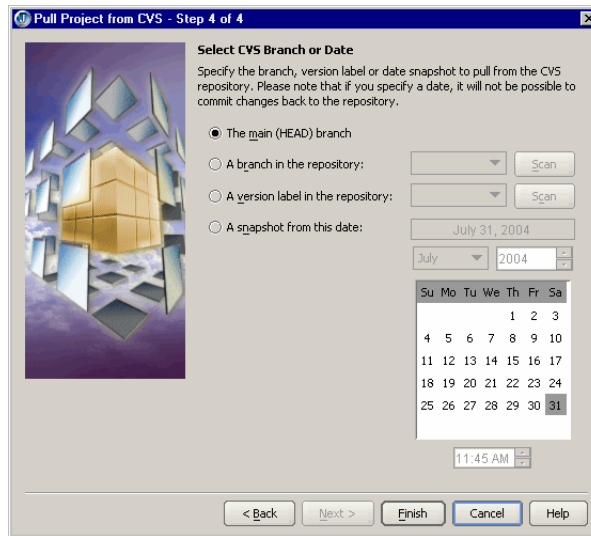
**Note**

If you selected Export in the first step of the Pull Project From CVS wizard, the Main (HEAD) Branch option is disabled. CVS requires you to specify a branch, version label, or date when you export files.

- The Main (HEAD) Branch: this pulls the most recent versions of the files available in the repository. This is the default selection.
- A Branch In The Repository: this lets you pull the project's files from a specific branch. CVS allows you to isolate changes onto a separate line of development,

known as a branch. When you change files on a branch, those changes do not appear on the main trunk or other branches.

- **A Version Label In The Repository:** this lets you pull versions of the project's files with a specific label (tag).
- **A Snapshot From This Date** lets you pull the most recent revision of the project's files no later than the specified date and time. If you use this option, you cannot commit changes to the files back to the repository.



- 13 Click Finish to complete the wizard.

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

- 14 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.

## Posting file 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.

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 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 CVSCommit "<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 and commands across the project

JBuilder provides tools which allow you to view the status of all the changed files in your project. Additional features let you determine which files to display in your workspace, which files to share, and which commands to run against multiple files.

**Table 10.1** CVS: Project-wide version control status, files, and commands

Tool	Show status	Set files	Run commands
Project pane decorations	X		
Status Browser	X	X	
Commit Browser	X	X	X
File Include Lists		X	

The Status Browser is a viewing tool. The Commit Browser incorporates the viewing features of the Status Browser, and adds context-aware CVS command functionality.

Access the Status Browser or Commit Browser in one of these ways:

- Choose Team|Status Browser or Team|Commit Browser.
- Right-click the project node and choose either CVS|Status Browser or CVS|Commit Browser.
- Right-click the project directory in the Files tab of the project pane and choose either CVS|Status Browser or CVS|Commit Browser.

These browsers have two pages:

- Changes/Commits page
- File Include Lists page

When you're done with the Status Browser, click OK to close it. It's purely a viewing tool, so nothing is changed.

When you're done with the Commit Browser, click the Commit button at the bottom:

- All of the commands you specified for the displayed files get executed.
- All of your comments are applied as specified.
- Files with No Action specified are left unchanged in relation to CVS.
- The Commit Browser closes and a progress dialog appears, displaying the progress of the version control operations.

If updating a file from the Commit Browser creates a merge conflict, the conflicts are clearly marked with “<<<<<<”, “=====”, and “>>>>>>”, and JBuilder’s merge conflict handling mechanisms are engaged.

### See also

- “Finding information and commands quickly” on page 66
- “Merge Conflicts page” in *Building Applications with JBuilder*

## Using the Changes and Commits pages

**In both Status  
Browser and Commit  
Browser**

These pages are very similar. The Status Browser uses the Changes page, which displays changes in files and directories. The Commit Browser uses the Commits page, which adds CVS commands and support for comments.

These pages consist of the following elements, starting in the upper left and going clockwise:

Element	Location	Status Browser	Commit Browser
Tree view	(Upper) left	X	X
File table	Upper right	X	X <sup>1</sup>
Summary Comment panel	Lower left		X
Tabbed source views	Lower right	X	X <sup>1</sup>

1. There is added functionality for this element in the Commit Browser.

### Tree view

**In both Status  
Browser and Commit  
Browser**

The tree view displays all of the changed files in their hierarchy. In the Commit Browser, it also provides access to commands which you can apply recursively, against every file below the level of the selected node.

To display this	Do this
All the files in the project (shown in the file table)	Select the Full List node in the directory tree
Subdirectories in the directory tree	Expand the module or parent directories
Files for an individual directory (shown in the file table)	Select the directory node in the directory tree
<b>Additionally, in the Commit Browser,</b>	
Apply CVS commands recursively	Right-click a node

### Table of changed files

**In both Status  
Browser and Commit  
Browser**

This table lists the changed files which belong to the node selected in the tree view on the left side of this page. Select a file in this table to view its source in the tabs below it in the dialog box.

<b>Action</b> <sup>1</sup>	Provides context-sensitive CVS commands.
<b>Status</b>	Displays the file’s current state in relation to CVS.
<b>File Name</b>	Displays the name of each file that has been changed in relation to CVS.

1. Only the Commit Browser uses the Action column. Both the Status Browser and the Commit Browser use the Status and File Name columns.

In the Commit Browser, the Action column provides CVS actions in a drop-down list for each file.

**Table 10.2** CVS Commit Browser: Version control options

Option	Description
Add	Add and commit this file so it's stored in the repository.
Commit	Check in the change to the repository.
Delete	This file is not in the repository. Delete removes it from your workspace, so your workspace matches the repository.
Get	This file has already been added to the repository; this checks it out to your workspace.
No Action	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.
Revert	Update the workspace with the latest repository version of this file, discarding all changes made since your last update.
Update	The file has changed in the repository; this retrieves the changes made to the file in the repository, and applies them to the file in the workspace.

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 Get.

Condition	Listed status	Default option	Alternative options
File changed in the workspace	Changed In Workspace	■ Commit	<ul style="list-style-type: none"> <li>■ Revert</li> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ No Action</li> </ul>
File added to version control	Not In Repository	■ Add	<ul style="list-style-type: none"> <li>■ Delete</li> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ No Action</li> </ul>
File deleted from version control	Not In Workspace	<ul style="list-style-type: none"> <li>■ <i>If removed from within JBuilder:</i> Remove</li> <li>■ <i>If removed from outside JBuilder:</i> Get</li> </ul>	<ul style="list-style-type: none"> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ Revert</li> <li>■ No Action</li> </ul>

**Note** When you add or remove files from the Commit Browser, they're automatically committed by JBuilder. When you add or remove files individually from a menu, they need to be committed in a separate step.

If you're solely interested in the Status Browser, skip ahead to [“Tabbed source views” on page 76](#).

## Summary Comment

In the Commit  
Browser

The Summary Comment panel is where you write a comment you want to apply to more than one file. To use it

- 1 Write a comment in this panel which applies to files in the file table.
- 2 As you go through the files in the table (upper right), check the Include Summary Comment check box at the bottom of the Individual Comment page (in the tabbed view) for the files you want to apply this summary comment to.
- 3 Optionally, write individual file comments under the Individual Comment tab.

Individual comments are appended to summary comments.

When you click the Commit button at the bottom of the page, JBuilder runs your CVS commands on those files and applies all comments as specified.

The Include Summary Comment check box at the bottom of the Individual Comment page allows you to specify whether or not to include the summary comment for individual files. This option is on by default, or when the individual comment is blank. If you uncheck Include Summary Comment for a file, only the individual comment will be applied when the file is committed.

When a summary comment is included, it appears before the individual comment, and both are maintained as a single comment for the revision. Summary comments are entered in the Summary Comment pane in the Commits page.

## Tabbed source views

In both Status  
Browser and Commit  
Browser

Select a file from the file table above to see its source and diffs displayed here. The appropriate source views for the file's CVS status become available. For instance, if the selected file was changed in the repository, the Workspace Source, Repository Source, and Repository Diff tabbed views become available.

**Table 10.3** CVS: Source views

Source view	Contents
Workspace Source	This file's source code from the current workspace version.
Repository Source	This file's source code from the current repository version.
Workspace Diff	This file's most recent changes in your workspace.
Repository Diff	This file's most recent changes in the repository.
Complete Diff	Differences between the current version of this file in the repository and the current version in your workspace.
<b>The Commit Browser adds one more tab:</b>	
Individual Comment	Enter your comments describing the version control action on the file selected in the file table above these tabs.

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.

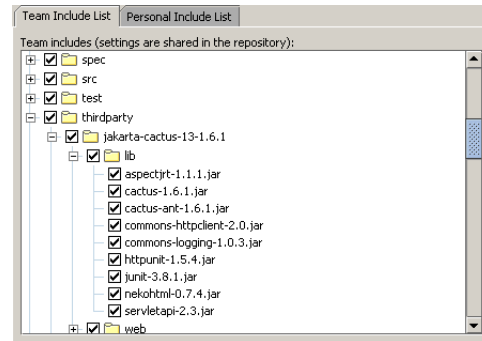
The Include Summary Comment check box is used to determine whether the summary comment should be used, either as the sole comment or else prepended to the individual comment written for this file. This is described in ["Summary Comment" on page 76](#).

## File Include Lists

In both Status  
Browser and Commit  
Browser

The File Include Lists determine which files are shared and which files are visible in your workspace. These are available from several places:

- TeamFile Include Lists
- File Include Lists page of the Status Browser
- File Include Lists page of the Commit Browser



There are two pages: Team Include List and Personal Include List.

- 1 Choose TeamFile Include Lists.
- 2 Use the Team Include List to determine which files are shared.
- 3 Use the Personal Include List to determine which files you want to keep in sight.

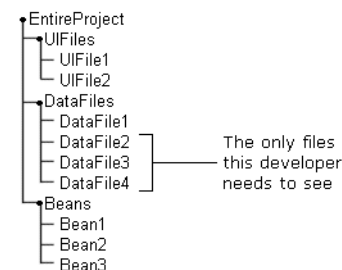
The files checked in the *Team Include List* are a team-wide project setting: the files that are checked here are the files that everyone needs to be able to use.

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 files checked in the *Personal Include List* are visible in your workspace; those that are unchecked are not, until you check them here. 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.

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



The Personal Include List page filters your view of the files, but does not affect the repository or the files themselves. Files that are checked on the Team Include List but not on the Personal Include List are still available on this page, and can be rechecked on the Personal Include List at any time.

## 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 not updated and committed with the rest of the project, but are handled separately. See [“Synchronizing project settings” on page 80](#).

### Updating a single file

---

When you update your workspace with changes from the repository, CVS automatically merges repository changes with your workspace 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 *merge conflict* and will not let the later user commit changes to that file.

When JBuilder discovers 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 concern, 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 97](#).

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.

### Reverting a file back to its checked out state

---

If you want to abandon any changes you may have made to the active file since you checked it out, you can choose Revert “<filename>” from the Team menu. This clears the editor buffer for the file, and returns it to the original state of the revision when it was checked out. This command does not affect the repository.

### 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 “>>>>>>>” to mark 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
// Current comment modified in file in the workspace
=====
// Last comment in file checked into the 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.

- In the history pane, JBuilder's Merge Conflicts page displays the workspace source, the repository source, and merge conflict handling tools to simplify the process of reconciliation.

**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.

The Status Browser dialog box (Team!Status Browser) provides another means of viewing differences between file revisions (diffs) 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.

**Important** The Merge Conflicts page is enabled only when JBuilder detects a conflict. Use CVS inside of JBuilder to enable merge assistance.

### See also

- "Using the history pane" in *Building Applications with JBuilder*
- "Merge Conflicts page" in *Building Applications with JBuilder* to learn how to use JBuilder's merge conflict handling mechanisms

## Reconciling conflicts manually

If it doesn't make sense to use the automatic merge conflict handling mechanisms, you can edit files manually to resolve conflicts.

To reconcile conflicts manually,

- 1 Search for the merge conflict markup in the file source:

- "<<<<<<" marks the top of the newer version of that area in your workspace.
- "=====" marks the boundary between the older version and newer version.
- ">>>>>>" marks the bottom of the older version in the repository.

For instance,

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

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

For instance,

```
<<<<<< Frame1.java
// Current comment modified in file in the workspace
=====
>>>>>> 1.3
```

3 Remove the conflict markup and any extraneous text.

For instance,

```
// Current comment modified in file in the workspace
```

4 Save the file, update, and commit the changes.

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

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

**Tip** The compiler will report errors if it finds conflict markup 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.

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

## Updating projects or project groups

---

These commands update 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.

To use the project update command, do one of the following

- Choose Team!Update Project
- In the project panelProject tab, right-click the project (.jpx) node and choose CVS! Update project

To update the project group

- Choose Team!Update Project Group

When you update your project in JBuilder, conflicts are flagged and JBuilder's conflict handling mechanisms are engaged. This can make it much easier and faster to resolve merge conflicts.

### See also

- ["Reconciling CVS merge conflicts" on page 78](#)
- "Merge Conflicts page" in *Building Applications with JBuilder*

## Synchronizing project settings

---

Each JBuilder project is administered by a project file. 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. This allows you to control when and whether global settings get posted to the repository.

- 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.



**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. Child project files are updated with the rest of the project.

To maintain the project file in version control

1 Open the Sync Project Settings submenu, available from either of these places:

- Team|Sync Project Settings
- Right-click the project file in the project node and choose CVS|Sync Project Settings.

2 Pull or post the project file:

- Sync Project Settings|Pull Latest "<projectfile>.jpx".
- Sync Project Settings|Post Current "<projectfile>.jpx".

The dialog box for pulling or posting the project file appears.

3 Click OK to perform the operation on the project file.

4 Click OK to close the dialog box.

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.

#### See also

- "Creating and managing projects" in *Building Applications with JBuilder* to understand project files better
- ["Pulling or posting a project file" on page 95](#)

## 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.
- 4 Commit the file to CVS using JBuilder.

**Note** CVS keeps the Add and Commit commands separate. This allows you to work with the addition locally before deciding whether to commit it to the repository.

To add an active file in the JBuilder project to the CVS module,

- 1 Choose Team|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.
- 5 Commit the file to CVS using JBuilder.

To add an inactive file or multiple files in the JBuilder project 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|Add Files from the context menu.  
If you are adding a single file, the format of the menu option will be CVS|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.
- 6 Commit the files to CVS using JBuilder.

**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.
- 5 Commit the removal to CVS using JBuilder.

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.
- 6 Commit the removal to CVS using JBuilder.

## Working with labels and branches

---

Labels and branches are important concepts for maintaining projects in version control systems. The following sections describe how to work with CVS labels (tags) and branches in JBuilder.

### Using version labels

---

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.

JBuilder supports `tag` and `rtag`. `tag` requires a local version of each file being labeled. `rtag` does not; it applies the label directly to the files in the repository.

To add a version label,

- 1 Choose Team|CVS Administration|Create Version Label.  
The Create Version Label dialog box appears.
- 2 Optionally, click Scan to scan the repository for any existing labels.
- 3 Type the name of the label you want to create in the Label Name text box.
- 4 Choose whether to use `tag` or `rtag`.

**Note** When using `tag`, if you have modified files in your workspace, CVS will not create the tag unless you check the Execute Even If Files Are Modified In Workspace check box.

- 5 Click OK to create the label.

For each file in the project under CVS version control, the label (tag) is applied to the most recent revision of the file in the repository.

To delete a version label,

- 1 Choose Team|CVS Administration|Delete Version Label.  
The Delete Version Label dialog box appears.
- 2 Click Scan to scan the repository for any existing labels.  
This populates the Version Label To Delete drop-down list.
- 3 Select the label you want to delete from the Version Label To Delete drop-down list.
- 4 Choose whether to use `tag` or `rtag`.

**Note** When using `tag`, if you have modified files in your workspace, CVS will not delete the tag unless you check the Execute Even If Files Are Modified In Workspace check box.

- 5 Click OK to delete the label.

For each file in the project under CVS version control, the label (tag) is removed from the revision of the file in the repository to which it was applied.

To move a version label,

- 1 Choose Team|CVS Administration|Move Version Label.  
The Move Version Label dialog box appears.
- 2 Click Scan to scan the repository for any existing labels.  
This populates the Version Label To Move drop-down list.

- 3 Select the label you want to move from the Version Label To Delete drop-down list.
- 4 Choose whether to use `tag` or `rtag`.

**Note** If you have modified files in your workspace, CVS will not move the version label unless you check the Execute Even If Files Are Modified In Workspace check box.

- 5 Click OK to move the label.

For each file in the project under CVS version control, the label (tag) is moved to the most recent revision of the file in the repository.

Version labels can be used as criteria for pulling a project from CVS. For more information on pulling a project for a specific label, see [“Checking out an existing project” on page 70](#).

## Creating a branch

---

CVS allows you to isolate changes onto a separate line of development, known as a branch. When you change files on a branch, those changes do not appear on the main trunk or other branches.

To create a branch,

- 1 Choose Team|CVS Administration|Create Branch.

The Create Branch dialog box appears.

- 2 Optionally, click Scan to scan the repository for any existing branch names.
- 3 Type the name of the branch you want to create in the Branch Name text box.

**Note** If you have modified files in your workspace, CVS will not create the branch unless you check the Create Branch Even If Files Are Modified In Workspace check box.

- 4 Click OK to create the branch.

For each file in the project under CVS version control, the label (tag) is applied to the most recent revision of the file in the repository.

Branches can be used as criteria for pulling a project from CVS, or for updating or merging a project in your workspace. For more information on pulling a project for a specific branch, see [“Checking out an existing project” on page 70](#).

## Special updates and merges

---

The Update Special and Merge commands provide the ability to capture changes associated with branches, labels, or dates, and combining them into the project in your workspace.

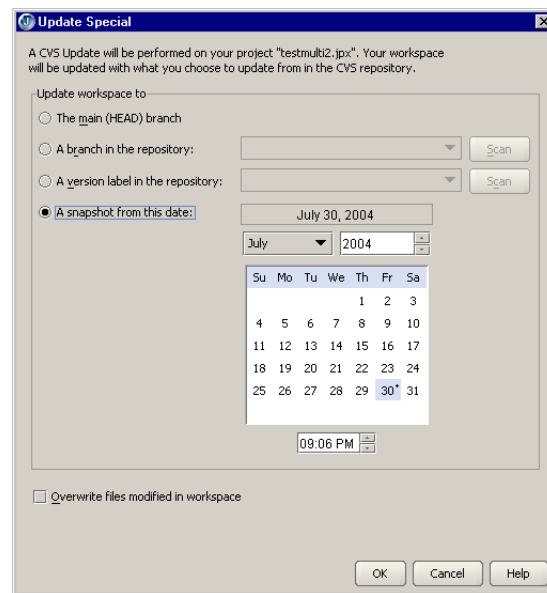
The Update Special command (Team|CVS Administration|Update Special) lets you update the project's files in your workspace with revisions in the repository based on branches, labels, or a date and time. This update removes sticky tags associated with the revisions being merged into your workspace files.

When you are done working on a branch, you can use the Merge command (Team|CVS Administration|Merge) to merge changes made on the branch into the project's files in your workspace. You can also merge changes to files with a specific version label into your workspace. You can then commit the workspace revision, and thus effectively copy the changes onto another branch or version label. This is similar to merging using the `-j` CVS command-line option.

To update the active project using special criteria,

- 1 Choose Team!CVS Administration!Update Special.

The Update Special dialog box appears:



- 2 Select one of the following criteria for the update:

- The Main (HEAD) Branch: this updates the project's files in your workspace with the most recent versions of the files available in the repository. This is the default selection.
- A Branch In The Repository: this lets you update the project's files in your workspace from a specific branch. Your updated files will not include the *sticky tag* of the branch you updated from, so you can commit your files.
- A Version Label In The Repository: this lets you update the project's files in your workspace with revisions tagged with a specific label (tag). Your updated files will not include the *sticky tag* for the label you updated from, so you can commit your files.
- A Snapshot From This Date: this lets update the project's files in your workspace from with the most recent revision of the project's files in the repository no later than the specified date and time. Your updated files will not include the *sticky tag* for the date you updated from, so you can commit your files.

**Note** If you have modified files in your workspace, CVS will not carry out the update operation unless you check the Overwrite Files Modified In Workspace check box.

- 3 If necessary, select the specific branch, label, or date.
- 4 Click OK to update your files.

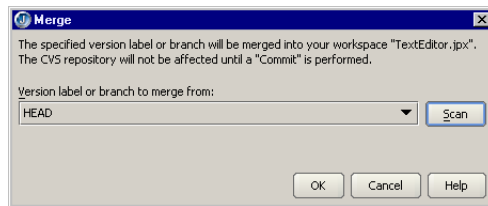
The Version Control Operation dialog box shows the progress of the update, and notifies you when the update is complete.

- 5 Click Close to close the Version Control Operation dialog box.

To merge a specific branch or labeled version of the project into your working files,

- 1 Choose Team|CVS Administration|Merge.

The Merge dialog box appears:



- 2 Click Scan to populate the Version Label Or Branch To Merge From drop-down list with existing branches and labels.
- 3 Click OK to merge the files.

The changes for the specified label or branch are merged into the active project's files in your workspace. The repository is not affected until you commit the files in your workspace.

## Tracking file status in CVS

---

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

### Checking a file's CVS status

---

This command 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.

The Project tab in the project pane indicates version control status for each file, by default. It does so by

- Adding a decorator to the icon for each changed file's node
- Providing a tool tip describing the status
- Adding descriptive remarks in square brackets to the node

It updates information from the repository every 15 minutes, by default.

Customize these settings in the Tools|Preferences dialog box, on the Project tab's Decorations page.

If you don't want to use alterations in the project pane to indicate file status, you can still check the status of your files individually using the following commands.

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.
- 3 Choose CVS|Status For "<filename>" from the context menu.
- 4 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.

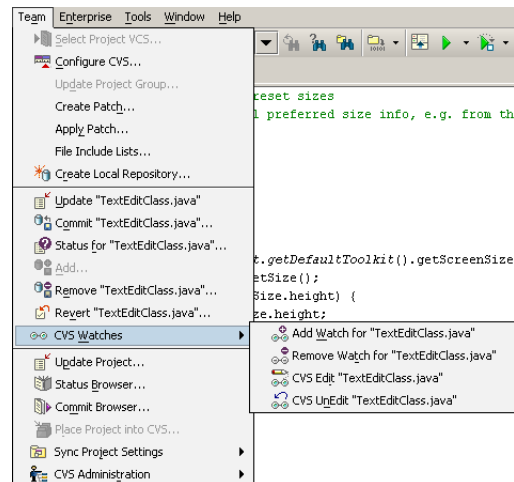
### CVS Watches

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:

- If you want to keep your changes, commit the file.

This will remove the CVS Edit, while leaving the Watch on the file.

- 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.



## Working on a new project in CVS

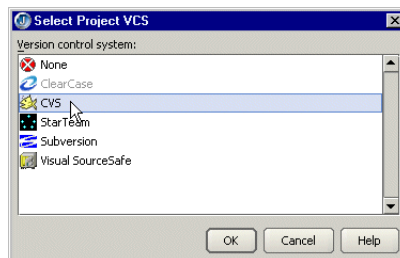
Once JBuilder is notified that you want to put the active project under version control, it populates the Team menu with applicable CVS commands. To notify JBuilder,

- 1 Open a project.

The open project will not be affected by CVS commands unless it's already under CVS management.

- 2 Choose Team|Select Project VCS.

The Select Project VCS dialog box appears:



- 3 Select CVS from the list of version control systems.
- 4 Click OK or press *Enter*.

This closes the dialog box and populates the Team menu with CVS commands.

## 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.

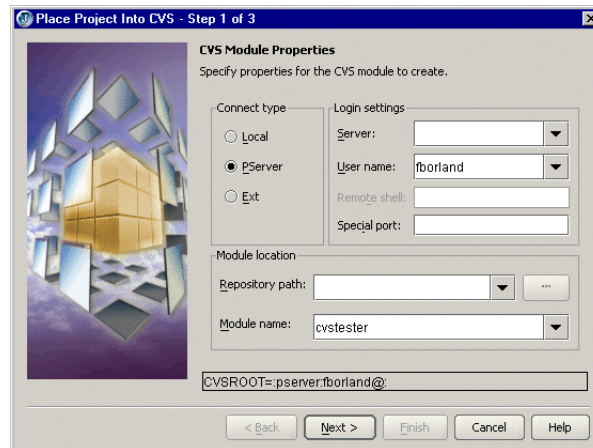
**Note** If the project you are placing into CVS is part of a project group, and the project group file (`.bpgx`) is in the root directory of the project, the project group file will also be checked in to CVS. Additionally, when the project is pulled from the CVS repository, the project group will also be pulled from the repository and opened.

## Checking in a project for the first time

A project needs to be checked in to CVS once, after which it can be maintained easily.

- 1 Open the project you want to check in.
- 2 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.

- 3 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.

- 4 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 can 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.

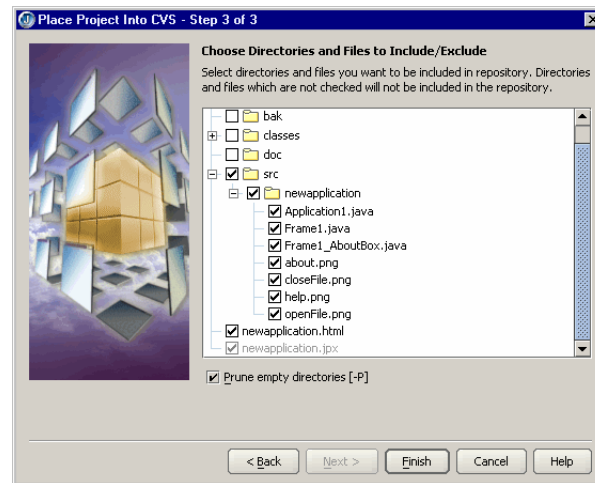
**Tip** The `CVSROOT` path appears at the bottom of the wizard. It reflects the information entered in this step. If you're unsure what you should enter and you already have a CVS account on an existing server, check your machine's user environment variables or see your CVS administrator.

- 5 Click Next to continue.

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

- 6 Enter a descriptive comment and click Next to continue.

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



- 7 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.

- 8 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.

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.

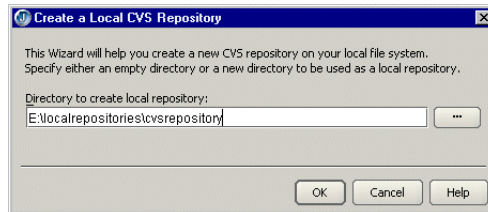
## Creating local repositories

JBuilder provides a way to create a local repository. A local repository gives you complete access and administrative control of the repository. This is ideal for working on projects that don't need to be shared but that would still benefit from revision management. It's also useful for managing subsets of larger projects, so you can work

out the bugs locally and avoid breaking the build when you check in to the shared server.

- 1 Open a project, to activate the Team menu.
- 2 Choose Team|CVS, to surface the CVS commands.
- 3 Choose Team|Create Local Repository.

The Create A Local CVS Repository dialog box appears:



- 4 Type in or browse to the path for the repository you want to create.

The target directory must be absolutely empty. At the end of the path, type a name for a new directory to make JBuilder create one for you.

- 5 Click OK to close the dialog box and create 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 12

## 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.
- [CVS 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.

A file's comments can be viewed in the 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 to use the CVS set of Team menu commands. 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 the IDE. However, if you do so, then the IDE will have no way of being aware of CVS actions performed outside of its own environment.

The CVS connection is set by environment variables. If the connection is configured properly in your environment variables before you use the IDE'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 the IDE 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

The IDE 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

---

Material is retrieved from the repository by checking out.

### 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 the IDE. Access it from the Project page of the object gallery: choose File|New|Project and double-click 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 <module>` command.

In the IDE, 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 or posting a project file

When you update a project file, JBuilder passes `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.

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 `<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 `cvs checkout <filename>` (and `cvs commit`) 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 `cvs update <filename1>, <filename2>, ...` to the command line at the working directory level.

When you update files from the Commit Browser, JBuilder puts the `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

The IDE 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 through the IDE.

## 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.

## 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 supports both `tag` and `rtag`. It passes `cvs tag|rtag <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 CVS-controlled files in its subdirectories.

`rtag` labels the repository files directly. `tag` requires a workspace revision, although the label is applied in the repository. CVS requires that you have your `CVSROOT` environment variable set for `rtag` to work.

### 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.

The online CVS manual discusses CVS watches at [https://www.cvshome.org/scdocs/ddUsingCVS\\_command-line.html.en#cvswatch](https://www.cvshome.org/scdocs/ddUsingCVS_command-line.html.en#cvswatch).

## 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, or Windows 2000.

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

- `cv`s: 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.
- `cv`sroot: 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:

Connection type	CVSROOT syntax
Local	<code>:local:&lt;repository path&gt;</code>
PServer	<code>:pserver:&lt;user name&gt;@&lt;server name&gt;:&lt;repository path&gt;</code>
Ext	<code>:ext:&lt;user name&gt;@&lt;server name&gt;:&lt;repository path&gt;</code>

### 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 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 TeamPlace 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.



# Chapter 13

## Version control reference: CVS

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

### Version control system API

---

The JBuilder IDE includes a framework for integrating any version control system (VCS), and using the VCS from within the IDE to manage projects. This capability is described by the VCS API, which you can use to customize an existing integration, or to design your own.

We recommend you start by reading “Introduction to PrimeTime and JBuilder OpenTools,” and “OpenTools basics” in *Developing OpenTools* to familiarize yourself with the structure and use of the OpenTools API documentation.

To learn how to integrate your own version control tool into the IDE, or customize the existing integrations, refer to “Version control system concepts” in *Developing OpenTools*, and the `com.borland.primetime.teamdev.frontend` and `com.borland.primetime.teamdev.vcs` packages.

#### See also

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

### 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).

## CVS resources

---

The following third-party resources provide additional information about the Concurrent Versions System (CVS).

- **List of newsgroups:** <http://www.cvshome.org/communication.html>
- **CVS Home:** <http://www.cvshome.org/>
- **CVS documentation:** <http://www.cvshome.org/docs/manual/index.html>
- **User information:** <http://www.devguy.com/fp/cfgmngmt/cvs/>

Special considerations:

- **Get CVS for various platforms from the source:** <http://www.cvshome.org/dev/codes.html>
- **CVS for the Mac (see the OpenTools documentation to extend JBuilder's capabilities):** <http://www.maccvs.org/> (home page)

Part III

# Using Subversion with JBuilder





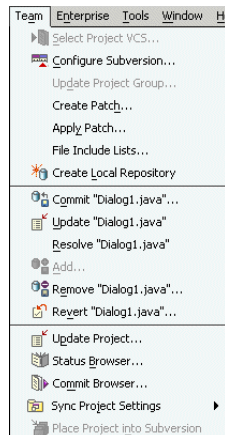
# Chapter 14

## Subversion in JBuilder

Subversion, an open source version control system, is integrated with all editions of JBuilder. JBuilder has a responsive interface that provides context-sensitive access to the most common Subversion commands from within the IDE.

Subversion is not installed by JBuilder; to use this integration, either choose Team | Configure Subversion and point to your installation, or make sure the Subversion `bin` directory is on the system path before launching JBuilder.

You can create modules, check out projects, add or remove files, check the Subversion status of any file or project, and commit changes from within the JBuilder interface, all from within the JBuilder IDE.



**Note** Subversion grew out of the Concurrent Versions System (CVS) project, but there are important differences between Subversion and CVS. This documentation assumes you are familiar with Subversion concepts and, if you also use CVS, are familiar with the differences between the two systems.

When a project is in Subversion, choose Team | Configure Subversion to view its connection configuration. This configuration can also be viewed in your project file settings.

### See also

- *Version Control with Subversion* at <http://svnbook.red-bean.com/>

## Finding information and commands quickly

---

JBuilder provides access to relevant version control information and commands from several convenient places, supporting many different working styles.

The history view in the content pane lets you examine prior revisions, look at diffs, read revision lists and comments, view checkins line-by-line, and handle merge conflicts if they arise.

In the project pane and the file tabs, JBuilder decorates the icons of files changed in version control. It provides textual notations and tool tips in the project pane describing the changed file's state. It checks the repository state every 15 minutes by default. Customize these settings in Tools|Preferences, on the Project tab's Decorations page.

Pertinent version control commands are available in these places:

- Choose the Team menu on the main menu bar for a complete list of commands.
- In the project pane, right-click these nodes:
  - Project node, in the Project tab
  - File node, in the Project tab
  - Any directory that's part of the project, in the File tab
- To pull a project from the repository, choose File|New|Project.
- To update the entire project recursively, type `jbuilder -update <projectname.jpx>` at the command line.

**Note** The update command is the only version control command available from the command line. It executes the project VCS's update command against every file in the project.

# Chapter 15

## Working on an existing project in Subversion

A JBuilder project must be under Subversion control before Subversion menu commands become available. This means you must either pull a project from Subversion or place an existing project into Subversion. Pulling an existing project is the more common task.

To work on an existing project, 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 Subversion 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.


**Tip** In the project pane, JBuilder decorates the icons of files changed in version control. Customize these settings in Tools|Preferences, Project tab, Decorations page.

**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.

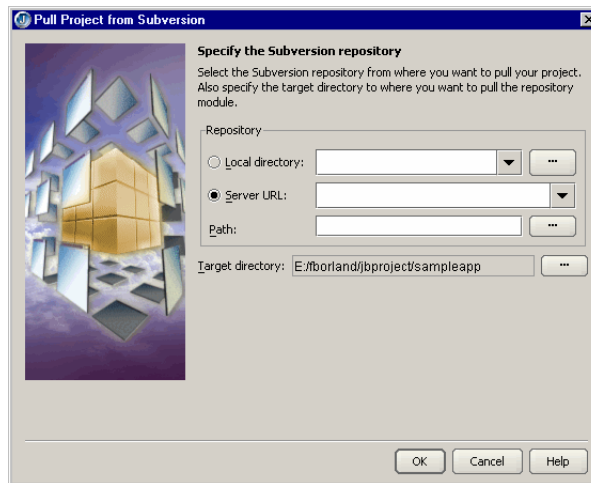
### Checking out an existing project

---

To check out a project that's already in the repository, use the Pull Project From Subversion wizard available in the object gallery:

- 1 Choose File|New|Project to open the Project page in the object gallery.
- 2 Select Pull Project From Subversion  .

- 3 Either click OK, double-click the icon, or press *Enter* to launch the Pull Project From Subversion wizard:



- 4 Indicate the repository you need:
  - **Local Directory**  
Type in or browse to your local repository's location. You can also choose from locations you've entered before.
  - **Server URL**  
Type in or choose the location of your server repository.  
Prior locations are available from the drop-down list.
- 5 Optionally, in the Path option, type in or browse to the particular branch or subdirectory you want to pull.
- 6 Click Finish to complete the wizard.  
The Checking Out Subversion Project dialog box appears and reports on the progress of the checkout.
- 7 Click OK to close the dialog box.

**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.

## Posting file changes

When you post changes, the VCS applies the altered file to the repository. When other users retrieve that file from the repository, they get the version that includes the changes you made. Posting changes is called either *committing* or *checking in*; these terms are used interchangeably here.

After changing a file in your workspace, use the Commit command to update the repository.

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

- 1 Choose Team|Commit "<filename>".  
The 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 Preferences dialog box (Tools|Preferences|Browser).

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

- 1 Right-click the file in the project pane.
- 2 Choose Subversion|Commit "<filename>".  
The Subversion 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 and increments the revision number of the repository tree.

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 and commands across the project

JBuilder provides tools which allow you to view the status of all the changed files in your project. Additional features let you determine which files to display in your workspace, which files to share, and which commands to run against multiple files.

**Table 15.1** Subversion: Project-wide version control status, files, and commands

Tool	Show status	Set files	Run commands
Project pane decorations	X		
Status Browser	X	X	
Commit Browser	X	X	X
File Include Lists		X	

The Status Browser is a viewing tool. The Commit Browser incorporates the viewing features of the Status Browser, and adds context-aware Subversion command functionality.

Access the Status Browser or Commit Browser in one of these ways:

- Choose Team|Status Browser or Team|Commit Browser.
- Right-click the project node and choose either Subversion|Status Browser or Subversion|Commit Browser.
- Right-click the project directory in the Files tab of the project pane and choose either Subversion|Status Browser or Subversion|Commit Browser.

These browsers have two pages:

- Changes/Commits page
- File Include Lists page

When you're done with the Status Browser, click OK to close it. It's purely a viewing tool, so nothing is changed.

When you're done with the Commit Browser, click the Commit button at the bottom:

- All of the commands you specified for the displayed files get executed.
- All of your comments are applied as specified.

- Files with No Action specified are left unchanged in relation to Subversion.
- The Commit Browser closes and a progress dialog appears, displaying the progress of the version control operations.

If updating a file from the Commit Browser creates a merge conflict, the conflicts are clearly marked with “<<<<<<”, “=====”, and “>>>>>>”, and JBuilder’s merge conflict handling mechanisms are engaged.

### See also

- “Finding information and commands quickly” on page 106
- “Merge Conflicts page” in *Building Applications with JBuilder*

## Using the Changes and Commits pages

In both Status  
Browser and Commit  
Browser

These pages are very similar. The Status Browser uses the Changes page, which displays changes in files and directories. The Commit Browser uses the Commits page, which adds Subversion commands and support for comments.

These pages consist of the following elements, starting in the upper left and going clockwise:

Element	Location	Status Browser	Commit Browser
Tree view	(Upper) left	X	X
File table	Upper right	X	X <sup>1</sup>
Summary Comment panel	Lower left		X
Tabbed source views	Lower right	X	X <sup>1</sup>

1. There is added functionality for this element in the Commit Browser.

### Tree view

In both Status  
Browser and Commit  
Browser

The tree view displays all of the changed files in their hierarchy. In the Commit Browser, it also provides access to commands which you can apply recursively, against every file below the level of the selected node.

To display this	Do this
All the files in the project (shown in the file table)	Select the Full List node in the directory tree
Subdirectories in the directory tree	Expand the module or parent directories
Files for an individual directory (shown in the file table)	Select the directory node in the directory tree
<b>Additionally, in the Commit Browser,</b>	
Apply Subversion commands recursively	Right-click a node

### Table of changed files

In both Status  
Browser and Commit  
Browser

This table lists the changed files which belong to the node selected in the tree view on the left side of this page. Select a file in this table to view its source in the tabs below it in the dialog box.

<b>Action</b> <sup>1</sup>	Provides context-sensitive Subversion commands.
<b>Status</b>	Displays the file’s current state in relation to Subversion.
<b>File Name</b>	Displays the name of each file that has been changed in relation to Subversion.

1. Only the Commit Browser uses the Action column. Both the Status Browser and the Commit Browser use the Status and File Name columns.

In the Commit Browser, the Action column provides Subversion actions in a drop-down list for each file.

**Table 15.2** Subversion Commit Browser: Version control options

Option	Description
Add	Add and commit this file so it's stored in the repository.
Commit	Check in the change to the repository.
Delete	This file is not in the repository. Delete removes it from your workspace, so your workspace matches the repository.
Get	This file has already been added to the repository; this checks it out to your workspace.
No Action	This changed file will not be touched by a Subversion operation of any kind. It will be exactly as you left it before you invoked the Commit Browser.
Revert	Update the workspace with the latest repository version of this file, discarding all changes made since your last update.
Update	The file has changed in the repository; this retrieves the changes made to the file in the repository, and applies them to the file in the workspace.

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 Get.

Condition	Listed status	Default option	Alternative options
File changed in the workspace	Changed In Workspace	■ Commit	<ul style="list-style-type: none"> <li>■ Revert</li> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ No Action</li> </ul>
File added to version control	Not In Repository	■ Add	<ul style="list-style-type: none"> <li>■ Delete</li> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ No Action</li> </ul>
File deleted from version control	Not In Workspace	<ul style="list-style-type: none"> <li>■ <i>If removed from within JBuilder: Remove</i></li> <li>■ <i>If removed from outside JBuilder: Get</i></li> </ul>	<ul style="list-style-type: none"> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ Revert</li> <li>■ No Action</li> </ul>

**Note** When you add or remove files from the Commit Browser, they're automatically committed by JBuilder. When you add or remove files individually from a menu, they need to be committed in a separate step.

If you're solely interested in the Status Browser, skip ahead to ["Tabbed source views" on page 112](#).

## Summary Comment

In the Commit  
Browser

The Summary Comment panel is where you write a comment you want to apply to more than one file. To use it

- 1 Write a comment in this panel which applies to files in the file table.
- 2 As you go through the files in the table (upper right), check the Include Summary Comment check box at the bottom of the Individual Comment page (in the tabbed view) for the files you want to apply this summary comment to.
- 3 Optionally, write individual file comments under the Individual Comment tab.

Individual comments are appended to summary comments.

When you click the Commit button at the bottom of the page, JBuilder runs your Subversion commands on those files and applies all comments as specified.

The Include Summary Comment check box at the bottom of the Individual Comment page allows you to specify whether or not to include the summary comment for individual files. This option is on by default, or when the individual comment is blank. If you uncheck Include Summary Comment for a file, only the individual comment will be applied when the file is committed.

When a summary comment is included, it appears before the individual comment, and both are maintained as a single comment for the revision. Summary comments are entered in the Summary Comment pane in the Commits page.

## Tabbed source views

In both Status  
Browser and Commit  
Browser

Select a file from the file table above to see its source and diffs displayed here. The appropriate source views for the file's CVS status become available. For instance, if the selected file was changed in the repository, the Workspace Source, Repository Source, and Repository Diff tabbed views become available.

**Table 15.3** Subversion: Source views

Source view	Contents
Workspace Source	This file's source code from the current workspace version.
Repository Source	This file's source code from the current repository version.
Workspace Diff	This file's most recent changes in your workspace.
Repository Diff	This file's most recent changes in the repository.
Complete Diff	Differences between the current version of this file in the repository and the current version in your workspace.
<b>The Commit Browser adds one more tab:</b>	
Individual Comment	Enter your comments describing the version control action on the file selected in the file table above these tabs.

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 Include Summary Comment check box is used to determine whether the summary comment should be used, either as the sole comment or else prepended to the individual comment written for this file. This is described in ["Summary Comment" on page 112](#).

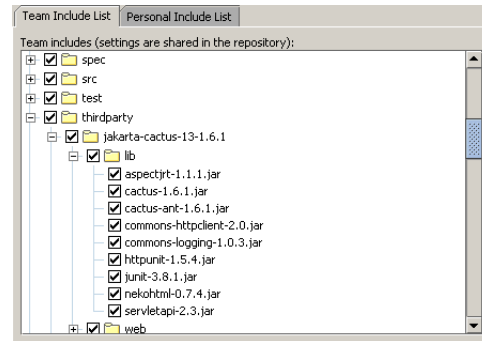


## File Include Lists

In both Status  
Browser and Commit  
Browser

The File Include Lists determine which files are shared and which files are visible in your workspace. These are available from several places:

- TeamFile Include Lists
- File Include Lists page of the Status Browser
- File Include Lists page of the Commit Browser



There are two pages: Team Include List and Personal Include List.

- 1 Choose TeamFile Include Lists.
- 2 Use the Team Include List to determine which files are shared.
- 3 Use the Personal Include List to determine which files you want to keep in sight.

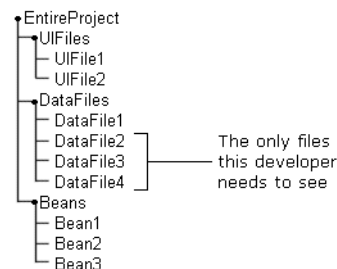
The files checked in the *Team Include List* are a team-wide project setting: the files that are checked here are the files that everyone needs to be able to use.

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 Subversion in JBuilder.

The files checked in the *Personal Include List* are visible in your workspace; those that are unchecked are not, until you check them here. 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.

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



The Personal Include List page filters your view of the files, but does not affect the repository or the files themselves. Files that are checked on the Team Include List but not on the Personal Include List are still available on this page, and can be rechecked on the Personal Include List at any time.

## Synchronizing with changes in the repository

*Updating* pulls the tip repository version into your workspace. This keeps your workspace current with changes made by others.

**Note** Project files (.jpx) are not processed with the rest of the files in the project. Update project files individually.

## Updating a single file

When you update your workspace with changes from the repository, JBuilder automatically merges repository changes with your workspace 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, the VCS registers a *merge conflict* and will not let the later user commit changes to that file.

When JBuilder discovers 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** Subversion 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 concern, when incompatible or problematic programming elements are used in the same program. Subversion is not designed to handle logical evaluations of programs, only physical assessments of files.

Subversion is designed to handle text-based files. It has special mechanisms for dealing with binary files. See the Subversion documentation at <http://svnbook.red-bean.com/> to understand how this works.

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.

## Reverting a file back to its checked out state

If you want to abandon any changes you may have made to the active file since you checked it out, you can choose Revert "<filename>" from the Team menu. This clears the editor buffer for the file, and returns it to the original state of the revision that was checked out. This command does not affect the repository.

## Reconciling Subversion merge conflicts

JBUILDER helps avoid merge conflicts by requiring you to update when necessary before committing changes. If merge conflicts do occur, Subversion 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.
- Subversion inserts “<<<<<<”, “=====”, and “>>>>>>” to mark 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
// Current comment modified in file in the workspace
=====
// Last comment in file checked into the 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.

- In the history pane, JBuilder's Merge Conflicts page displays the workspace source, the repository source, and merge conflict handling tools to simplify the process of reconciliation.

There are two ways to reconcile Subversion merge conflicts in JBuilder: automatically or manually. Automatic merge conflict handling is best for straightforward block conflicts, and is done in the Merge Conflicts page of the history pane. Manual handling is best for more complicated conflicts, and is done by you in the source editor. Choose the way that best addresses the conflict.

When you have reconciled the merge conflict, choose Team|Resolve to notify Subversion and JBuilder of the reconciliation. Then you can update.

**Important** The Merge Conflicts page is enabled only when JBuilder detects a conflict. Use Subversion inside of JBuilder to enable merge assistance.

#### See also

- "Using the history pane" in *Building Applications with JBuilder*
- "Merge Conflicts page" in *Building Applications with JBuilder*

## Resolving conflicts

Subversion requires you to post a reconciliation to the repository after handling a merge conflict and before checking it in.

After you have reconciled a merge conflict (either automatically or manually), choose Team|Resolve to notify the repository that the merge conflict has been handled and the file can now be posted safely.

## 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 the "Merge Conflicts page" topic in *Building Applications with JBuilder*.

## Synchronizing project settings

---

Each JBuilder project is administered by a project file. 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. This allows you to control when and whether global settings get posted to the repository.

- 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.

**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. Child project files are updated with the rest of the project.

To maintain the project file in version control

- 1 Open the Sync Project Settings submenu, available from either of these places:
  - Team|Sync Project Settings
  - Right-click the project file in the project node and choose Subversion|Sync Project Settings.
- 2 Pull or post the project file:
  - Sync Project Settings|Pull Latest “<projectfile>.jpx”.
  - Sync Project Settings|Post Current “<projectfile>.jpx”.

The dialog box for pulling or posting the project file appears.

- 3 Click OK to perform the operation on the project file.
- 4 Click OK to close the dialog box.

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.

#### See also

- “Creating and managing projects” in *Building Applications with JBuilder* to understand project files better.

## Adding files

---

Adding a file to the JBuilder project is a separate step from adding it to the Subversion module. This command adds the file to the Subversion module.

A file that will be added to Subversion must be inside 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 Subversion is,

- 1 Put the file in the project directory.
- 2 Add the file to the JBuilder project in your workspace.
- 3 Add the file to the Subversion module.
- 4 Commit the file to the Subversion repository using JBuilder.

**Note** Subversion keeps the Add and Commit commands separate. This allows you to work with the addition locally before deciding whether to commit the file to the repository.

To add an active file in the JBuilder project to the Subversion module,

- 1 Choose Team|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.
- 5 Commit the file to Subversion using JBuilder.

To add an inactive file or multiple files in the JBuilder project to the Subversion 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 Subversion|Add Files from the context menu.  
If you are adding a single file, the format of the menu option will be Subversion|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.
- 6 Commit the files to Subversion using JBuilder.

**Caution** While files referred to in a JBuilder project can reside in different directory trees, files in a Subversion module *must reside in a single tree*. Therefore, before you add a file to Subversion, 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 Subversion within JBuilder removes the active file from the repository, from your workspace, and from the project.

To remove the active file from the Subversion 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.
- 5 Commit the removal to Subversion using JBuilder.

To remove an inactive file or multiple files from the Subversion 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 Subversion|Remove Files from the context menu.  
If you are removing a single file, the format of the menu option will be Subversion|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.
- 6 Commit the removal to Subversion using JBuilder.

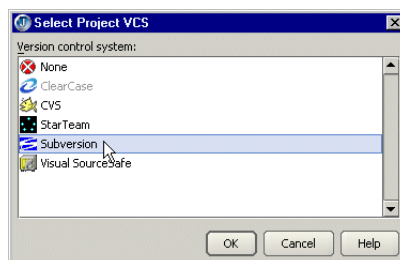
# Chapter 16

## Working on a new project in Subversion

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

- 1 Open a project.
- 2 Choose Team|Select Project VCS.

The Select Project VCS dialog box appears:



- 3 Select Subversion from the list of version control systems.
- 4 Click OK or press *Enter*.

This closes the dialog box and populates the Team menu with Subversion commands.

### Placing a new project into Subversion

---

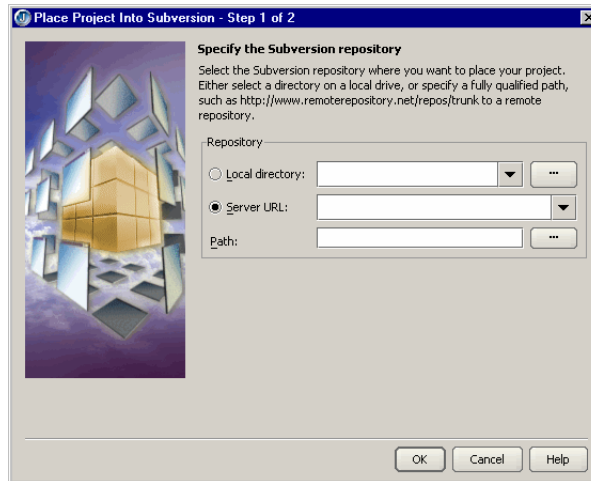
Checking a project into Subversion through JBuilder creates the directory structure in the repository, adds the files to the repository, and creates the Subversion infrastructure needed to maintain revision management both in the repository and in your workspace.

**Note** If the project you are placing into Subversion is part of a project group, and the project group file (`.bpgx`) is in the root directory of the project, the project group file will also be checked in to Subversion. Additionally, when the project is pulled from the Subversion repository, the project group will also be pulled from the repository and opened.

## Checking in a project for the first time

A project needs to be checked in to Subversion once, after which it can be maintained easily.

- 1 Open the project you want to check in.
- 2 Choose TeamPlace Project Into Subversion to bring up the Place Project Into Subversion wizard.

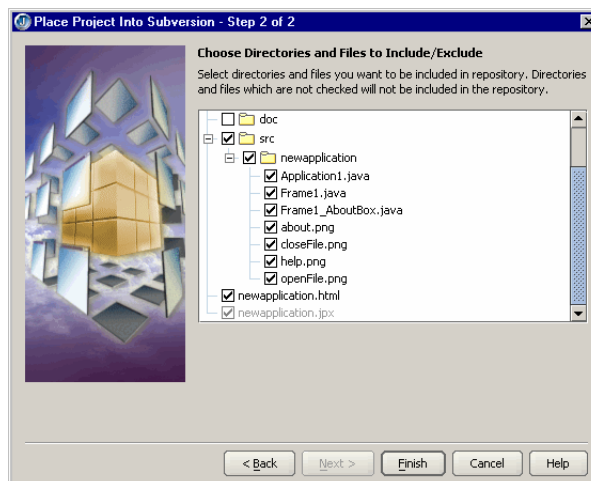


The first step of this wizard configures the connection to the repository.

Local Directory connects to a local repository and Server URL connects to a repository on a server.

- 3 Choose the repository type (Local Directory or Server URL) and type its location, browse to it, or choose it from prior entries.
- 4 Enter or browse to the path of the specific branch you want to use.
- 5 Click Next to specify which files and directories are included in the repository.
- 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 Subversion 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 Subversion control.

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

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

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

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

## Creating local repositories

---

JBuilder provides a way to create a local repository. A local repository gives you complete access and administrative control of the repository. This is ideal for working on projects that don't need to be shared but that would still benefit from revision management. It's also useful for managing subsets of larger projects, so you can work out the bugs locally and avoid breaking the build when you check in to the shared server.

To create a local repository

- 1 Open a project, to activate the Team menu.
- 2 Choose Team|Select Project VCS and select Subversion.
- 3 Choose Team|Subversion, to surface the Subversion commands.
- 4 Choose Team|Create Local Repository.

The Create A Local Subversion Repository dialog box appears.

- 5 Type in or browse to the path for the repository you want to create.

The target directory must be absolutely empty. If you browse, then, at the end of the path, type a name for a new directory to make JBuilder create one for you.

- 6 Click OK to close the dialog box and create the repository.

Subversion's administrative subdirectory is automatically created in the repository directory.

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

**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 Subversion useless in that repository, since Subversion has no way of tracking changes that were not made through Subversion.



Part IV

# Using Visual SourceSafe with JBuilder



# Chapter 17

## Visual SourceSafe in JBuilder

**Visual SourceSafe  
integration is a  
feature of JBuilder  
Developer and  
Enterprise**

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 2000 and XP.

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. With the exception of the Visual SourceSafe Explorer, which launches the VSS Explorer, Team menu commands apply to the active file and project. The Visual SourceSafe context menu commands apply to the selected files 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

---

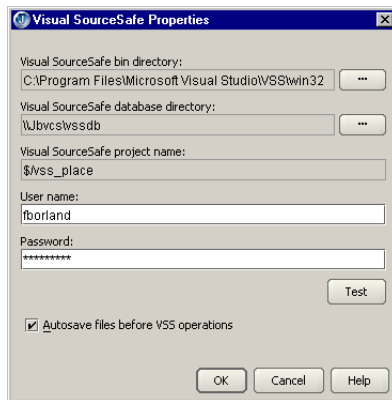
Place Project Into VSS (available from the Team menu) and Pull Project From VSS (available from the object gallery) are wizards that configure your connection for the project you want to work on. Each project's configuration is set individually.

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:



**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 145](#)

## Finding information and commands quickly

---

JBuilder provides access to relevant version control information and commands from several convenient places, supporting many different working styles.

The history view in the content pane lets you examine prior revisions, look at diffs, read revision lists and comments, view checkins line by line, and handle merge conflicts if they arise.

In the project pane and the file tabs, JBuilder decorates the icons of files changed in version control. It provides textual notations and tool tips in the project pane describing each changed file’s state. It checks the repository state every 15 minutes by default. Customize these settings in the Tools|Preferences dialog box, on the Project tab’s Decorations page.

Pertinent version control commands are available in these places:

- Choose the Team menu on the main menu bar for a complete list of commands.
- In the project pane, right-click these nodes:
  - Project node, in the Project tab
  - File node, in the Project tab
  - Any directory that’s part of the project, in the File tab
- To pull a project from the repository, choose File|New|Project.
- To update the entire project recursively, type `jbuilder -update <projectname.jpx>` at the command line.

**Note** The update command is the only version control command available from the command line. It executes Visual SourceSafe’s update command against every file in the project.

# Chapter 18

## Working on an existing project in VSS


Visual SourceSafe integration is a feature of JBuilder Developer and Enterprise

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

### Pulling an existing project

---

Pull a project from the database via the object gallery:

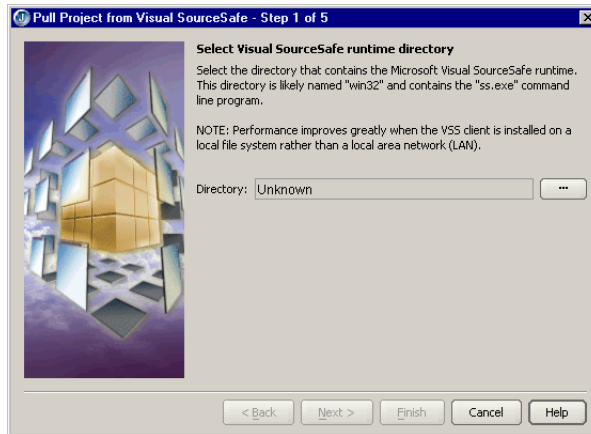
- 1 Choose File|New|Project to open the Project panel in the object gallery.
- 2 Select Pull Project From VSS .
- 3 Double-click the icon, click OK, or press *Enter* to start the Pull Project From Visual SourceSafe wizard.

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

This brings 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:



**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:

- 1 Select Visual SourceSafe Database Directory
- 2 Enter Username And Password
- 3 Select Visual SourceSafe Project
- 4 Select 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 tree view of projects in the Visual SourceSafe database for you to navigate. Branches can be expanded or collapsed. Select a Visual SourceSafe project from the tree. If the selected VSS project contains a JBuilder project (.jpx) file, the wizard opens the JBuilder project when the pull operation is complete. If the VSS project directory does not contain a JBuilder project file, the wizard creates a new JBuilder project when the pull operation is complete.

## Choosing a target directory

---

The Select Empty Target Directory step asks for an empty directory into which the project will be pulled. 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 145](#) 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, indicating the name of 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 you made since checking it out.

To undo the checkout of the active file,

- 1 Choose Team|Undo Check Out "<filename>".

An Undo Checkout dialog box appears, indicating the name of 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.

Once you undo a checkout, one of two things happens to your workspace:

- either the file version simply reverts to the version you originally checked out, or
- the latest database version of the file is pulled into your workspace.

This action depends on a Visual SourceSafe variable that your administrator sets. Talk to your Visual SourceSafe administrator if you have any questions.

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

### See also

- ["Undoing a checkout" on page 147](#)

## 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** When 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 Visual SourceSafe merge conflicts" on page 132](#) 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, indicating the name of 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.

**Note** If you want to abandon any changes you may have made to the active file since you checked it out, you can choose Revert "<filename>" from the Team menu. This clears the editor buffer for the file, and returns it to the original state of the revision when it was checked out. This command does not affect the repository.

## Synchronizing project settings

Each JBuilder project is administered by a project file. 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. This allows you to control when and whether global settings get posted to the database.

- 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.

**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. Child project files are updated with the rest of the project.

To maintain the project file in version control

- 1 Open the Sync Project Settings submenu, available from either of these places:
  - Team|Sync Project Settings
  - Right-click the project file in the project node and choose Visual SourceSafe|Sync Project Settings.
- 2 Pull or post the project file:
  - Sync Project Settings|Pull Latest "<projectfile>.jpx".
  - Sync Project Settings|Post Current "<projectfile>.jpx".

The dialog box for pulling or posting the project file appears.

- 3 Click OK to perform the operation on the project file.
- 4 Click OK to close the dialog box.

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.

#### 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 148](#)

## Reconciling Visual SourceSafe merge conflicts

---

JBuilder helps avoid merge conflicts by requiring you to update when necessary before committing changes.

If merge conflicts do occur, Visual SourceSafe and JBuilder alert you in the following ways:

- Visual SourceSafe inserts tags that mark the conflicting blocks in the file in your workspace:

```
<<<<<< Filename.java
workspace version
=====
database version
>>>>>> RepositoryVersionNumber
```

- 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.
- In the history pane, JBuilder’s Merge Conflicts page displays the workspace source, the database source, and merge conflict handling tools to simplify the process of reconciliation.

**Important** Once a merge conflict is tagged, Visual SourceSafe no longer sees it as a conflict.

**Note** The compiler reports errors if it finds conflict tags when it compiles.

There are two ways to reconcile Visual SourceSafe merge conflicts in JBuilder:

- 1 Reconcile them manually, as explained in the next topic, using an editor to resolve the conflicts.
- 2 Reconcile them automatically, using the features in the Merge Conflicts page in the history pane to resolve the conflicts.

Choose the way that best addresses the conflict.

**Important** The Merge Conflicts page is enabled only when JBuilder detects a conflict. Use Visual SourceSafe inside of JBuilder to enable merge assistance.

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

#### See also

- “Using the history pane” in *Building Applications with JBuilder*
- “Merge Conflicts page” in *Building Applications with JBuilder*
- [“Changes and commands across the project” on page 135](#) for more on the Status Browser

## Reconciling conflicts manually

---

If it doesn't make sense to use the automatic merge conflict handling mechanisms, you can edit files manually to resolve conflicts.

To reconcile conflicts manually,

1 Search for the merge conflict markup in the file source:

- “<<<<<<” marks the top of the newer version of that area in your workspace.
- “=====” marks the boundary between the older version and newer version.
- “>>>>>>” marks the bottom of the older version in the database.

For instance,

```
<<<<<< Frame1.java
// Current comment modified in file in the workspace
=====
// Last comment in file checked into the database
>>>>>> 1.3
```

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

For instance,

```
<<<<<< Frame1.java
// Combined comments from workspace and database
=====
>>>>>> 1.3
```

3 Remove the conflict tags and any extraneous text.

For instance,

```
// Combined comments from workspace and database
```

4 Save the file, update, and commit the changes.

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

**Note** Once conflicts have been flagged, Visual SourceSafe no longer sees them as conflicts.

**Tip** The compiler reports errors if it finds conflict tags when it compiles. Therefore, if you leave the conflict tags in, you can postpone resolving a conflict, then compile and double-click the error message to finish up.

### See also

- “Merge Conflicts page” in *Building Applications with JBuilder*

## 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, indicating the name of 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.

## Changes and commands across the project

JBuilder provides tools which allow you to view the status of all the changed files in your project. Additional features let you determine which files to display in your workspace, which files to share, and which commands to run against multiple files.

**Table 18.1** VSS: Project-wide version control status, files, and commands

Tool	Show status	Set files	Run commands
Project pane decorations	X		
Status Browser	X	X	
Commit Browser	X	X	X
File Include Lists		X	

The Status Browser is a viewing tool. The Commit Browser incorporates the viewing features of the Status Browser, and adds context-aware Visual SourceSafe command functionality.

Access the Status Browser or Commit Browser in one of these ways:

- Choose Team|Status Browser or Team|Commit Browser.
- Right-click the project node and choose either Visual SourceSafe|Status Browser or Visual SourceSafe|Commit Browser.
- Right-click the project directory in the Files tab of the project pane and choose either Visual SourceSafe|Status Browser or Visual SourceSafe|Commit Browser.

These browsers have two pages:

- Changes/Commits page
- File Include Lists page

When you're done with the Status Browser, click OK to close it. It's purely a viewing tool, so nothing is changed.

When you're done with the Commit Browser, click the Commit button at the bottom:

- All of the commands you specified for the displayed files get executed.
- All of your comments are applied as specified.
- Files with No Action specified are left unchanged in relation to Visual SourceSafe.
- The Commit Browser closes and a progress dialog appears, displaying the progress of the version control operations.

If updating a file from the Commit Browser creates a merge conflict, the conflicts are clearly marked with “<<<<<<”, “=====”, and “>>>>>>”, and JBuilder's merge conflict handling mechanisms are engaged.

### See also

- [“Finding information and commands quickly” on page 126](#)
- “Merge Conflicts page” in *Building Applications with JBuilder*

## Using the Changes and Commits pages

**In both Status  
Browser and Commit  
Browser**

These pages are very similar. The Status Browser uses the Changes page, which displays changes in files and directories. The Commit Browser uses the Commits page, which adds Visual SourceSafe commands and support for comments.

These pages consist of the following elements, starting in the upper left and going clockwise:

Element	Location	Status Browser	Commit Browser
Tree view	(Upper) left	X	X
File table	Upper right	X	X <sup>1</sup>
Summary Comment panel	Lower left		X
Tabbed source views	Lower right	X	X <sup>1</sup>

1. There is added functionality for this element in the Commit Browser.

## Tree view

In both Status  
Browser and Commit  
Browser

The tree view displays all of the changed files in their hierarchy. In the Commit Browser, it also provides access to commands which you can apply recursively, against every file below the level of the selected node.

To display this	Do this
All the files in the project (shown in the file table)	Select the Full List node in the directory tree
Subdirectories in the directory tree	Expand the module or parent directories
Files for an individual directory (shown in the file table)	Select the directory node in the directory tree
Additionally, in the Commit Browser,	
Apply Visual SourceSafe commands recursively	Right-click a node

## Table of changed files

In both Status  
Browser and Commit  
Browser

This table lists the changed files which belong to the node selected in the tree view on the left side of this page. Select a file in this table to view its source in the tabs below it in the dialog box.

<b>Action</b> <sup>1</sup>	Provides context-sensitive Visual SourceSafe commands.
<b>Status</b>	Displays the file's current state in relation to Visual SourceSafe.
<b>File Name</b>	Displays the name of each file that has been changed in relation to Visual SourceSafe.

1. Only the Commit Browser uses the Action column. Both the Status Browser and the Commit Browser use the Status and File Name columns.

In the Commit Browser, the Action column provides Visual SourceSafe actions in a drop-down list for each file.

**Table 18.2** VSS Commit Browser: Version control options

Option	Description
Add	Add this file so it's stored in the database.
Commit	Check in the change to the database.
Delete	This file is not in the database. Delete removes it from your workspace, so your workspace matches the database.
Get	This file has already been added to the database; this checks it out to your workspace.



**Table 18.2** VSS Commit Browser: Version control options (continued)

Option	Description
No Action	This changed file will not be touched by a Visual SourceSafe operation of any kind. It will be exactly as you left it before you invoked the Commit Browser.
Rename Back To "<orig_filename>"	This file has been renamed or moved in the workspace. When you click Commit, the file in the workspace will be renamed back to its original name or returned to its original location, and no action is performed on the file in the database.
Rename To "<new_filename>"	This file has been renamed or moved in the workspace. When you click Commit, the file in the database will be renamed to the new name or moved to the new folder, and no action is performed on the file in the workspace.
Undo Checkout	Update the workspace with the latest database version of this file, discarding all changes made since your last update.

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 Get.

Condition	Listed status	Default option	Alternative options
File changed in the workspace	Changed In Workspace	■ Commit	<ul style="list-style-type: none"> <li>■ Undo Checkout</li> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ No Action</li> </ul>
File added to version control	Not In Repository	■ Add	<ul style="list-style-type: none"> <li>■ Delete</li> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ No Action</li> </ul>
File deleted from version control	Not In Workspace	<ul style="list-style-type: none"> <li>■ <i>If removed from within JBuilder:</i> Remove</li> <li>■ <i>If removed from outside JBuilder:</i> Get</li> </ul>	<ul style="list-style-type: none"> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ Undo Checkout</li> <li>■ No Action</li> </ul>

**Note** When you add or remove files from the Commit Browser, they're automatically committed by JBuilder. When you add or remove files individually from a menu, they need to be committed in a separate step.

If you're solely interested in the Status Browser, skip ahead to ["Tabbed source views" on page 138](#).

## Summary Comment

In the Commit  
Browser

The Summary Comment panel is where you write a comment you want to apply to more than one file. To use it

- 1 Write a comment in this panel which applies to files in the file table.
- 2 As you go through the files in the table (upper right), check the Include Summary Comment check box at the bottom of the Individual Comment page (in the tabbed view) for the files you want to apply this summary comment to.
- 3 Optionally, write individual file comments under the Individual Comment tab.

Individual comments are appended to summary comments.

When you click the Commit button at the bottom of the page, JBuilder runs your Visual SourceSafe commands on those files and applies all comments as specified.

The Include Summary Comment check box at the bottom of the Individual Comment page allows you to specify whether or not to include the summary comment for individual files. This option is on by default, or when the individual comment is blank. If you uncheck Include Summary Comment for a file, only the individual comment will be applied when the file is committed.

When a summary comment is included, it appears before the individual comment, and both are maintained as a single comment for the revision. Summary comments are entered in the Summary Comment pane in the Commits page.

## Tabbed source views

In both Status  
Browser and Commit  
Browser

Select a file from the file table above to see its source and diffs displayed here. The appropriate source views for the file's CVS status become available. For instance, if the selected file was changed in the database, the Workspace Source, Repository Source, and Repository Diff tabbed views become available.

**Table 18.3** VSS: Source views

Source view	Contents
Workspace Source	This file's source code from the current workspace version.
Repository Source	This file's source code from the current database version.
Workspace Diff	This file's most recent changes in your workspace.
Repository Diff	This file's most recent changes in the database.
Complete Diff	Differences between the current version of this file in the database and the current version in your workspace.
<b>The Commit Browser adds one more tab:</b>	
Individual Comment	Enter your comments describing the version control action on the file selected in the file table above these tabs.

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.

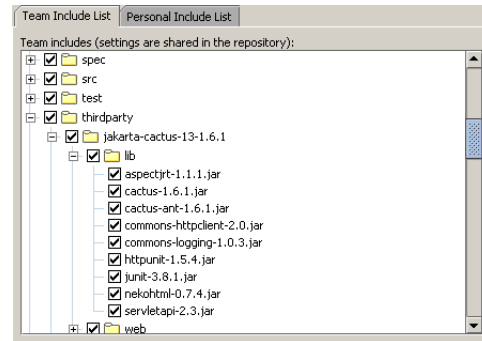
The Include Summary Comment check box is used to determine whether the summary comment should be used, either as the sole comment or else prepended to the individual comment written for this file. This is described in ["Summary Comment" on page 138](#).

## File Include Lists

**In both Status  
Browser and Commit  
Browser**

The File Include Lists determine which files are shared and which files are visible in your workspace. These are available from several places:

- TeamFile Include Lists
- File Include Lists page of the Status Browser
- File Include Lists page of the Commit Browser



There are two pages: Team Include List and Personal Include List.

- 1 Choose TeamFile Include Lists.
- 2 Use the Team Include List to determine which files are shared.
- 3 Use the Personal Include List to determine which files you want to keep in sight.

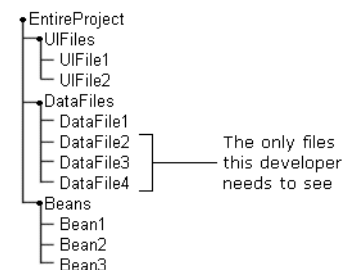
The files checked in the *Team Include List* are a team-wide project setting: the files that are checked here are the files that everyone needs to be able to use.

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 Visual SourceSafe in JBuilder.

The files checked in the *Personal Include List* are visible in your workspace; those that are unchecked are not, until you check them here. 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.

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



The Personal Include List page filters your view of the files, but does not affect the database or the files themselves. Files that are checked on the Team Include List but not on the Personal Include List are still available on this page, and can be rechecked on the Personal Include List at any time.

## 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 149](#) for usage guidelines

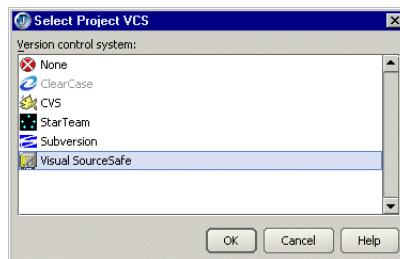
## Working on a new project in VSS

**Visual SourceSafe  
integration is a  
feature of JBuilder  
Developer and  
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:



- 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 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 TeamPlace Project Into VSS.

The Select Visual SourceSafe Database Directory page appears. After the first use, the first database you used appears 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.

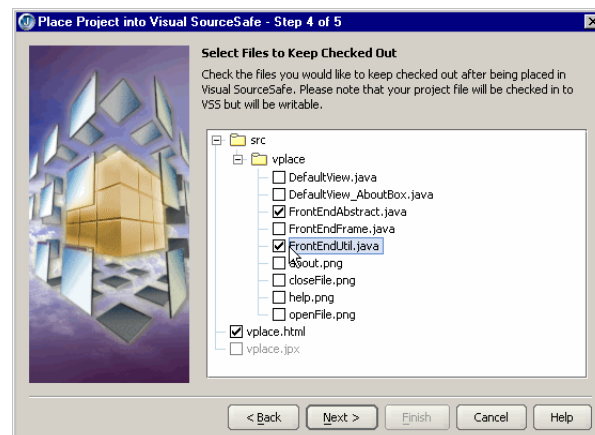
If your user name or password changes in Visual SourceSafe, you can change it to match by choosing TeamConfigure Visual SourceSafe.

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

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 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 into which you want to place the project.

All of the root directories available in the database are displayed in a tree, with branches that can be expanded and collapsed.

8 In the Project Name field, type a unique project name for this project.

This will be the project directory created inside the database's selected 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 127](#) to learn more about runtime directory discovery.





# Chapter 20

## Programmer's guide to the VSS integration

**Visual SourceSafe integration is a feature of JBuilder Developer and 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 146](#) 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 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 Select Visual SourceSafe Project step provides a tree view of projects in the Visual SourceSafe database for you to navigate. Branches can be expanded or collapsed. Select a Visual SourceSafe project from the tree. If the selected VSS project contains a JBuilder project (`.jpx`) file, the wizard opens the JBuilder project when the pull operation is complete. If the VSS project directory does not contain a JBuilder project file, the wizard creates a new JBuilder project when the pull operation is complete.

## 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 145](#) 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 `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 `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 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.

### See also

- [“Configuring the database connection” on page 145](#) to learn about the first steps of this wizard when it's first used.
- [“Selecting the Visual SourceSafe database directory” on page 146](#) 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 146](#) to learn about the first steps of this wizard when it's first used.

# Chapter 21

## Version control reference: Visual SourceSafe

Visual SourceSafe  
integration is a feature  
of JBuilder Developer  
and Enterprise

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

### Version control system API

---

The JBuilder IDE includes a framework for integrating any version control system (VCS), and using the VCS from within the IDE to manage Java projects. This capability is described by the VCS API, which you can use to customize an existing integration, or to design your own.

We recommend you start by reading “Introduction to PrimeTime and JBuilder OpenTools,” and “OpenTools basics” in *Developing OpenTools* to familiarize yourself with the structure and use of the OpenTools API documentation.

To learn how to integrate your own version control tool into the JBuilder IDE, or customize the existing integrations, refer to “Version control system concepts” in *Developing OpenTools* and the `com.borland.primetime.teamdev.frontend` and `com.borland.primetime.teamdev.vcs` packages.

#### See also

- [“Developer support and resources” on page 4](#) for information on helpful JBuilder and Borland links and newsgroups

## 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).

## Visual SourceSafe resources

---

The following resource provides additional information about Microsoft Visual SourceSafe.

- Home page: <http://msdn.microsoft.com/ssafe/>



P a r t   V

# Using ClearCase with JBuilder



## ClearCase in JBuilder

**ClearCase integration  
is a feature of  
JBuilder Developer  
and Enterprise**

JBuilder's integration of 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 Red Hat Enterprise Linux 3.0. 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.

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

## Selecting ClearCase as your version control system

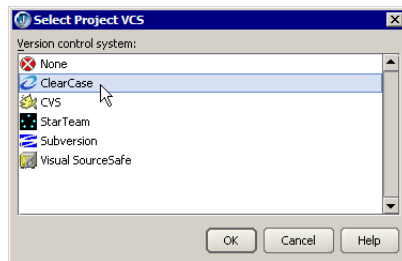
---

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 database 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:



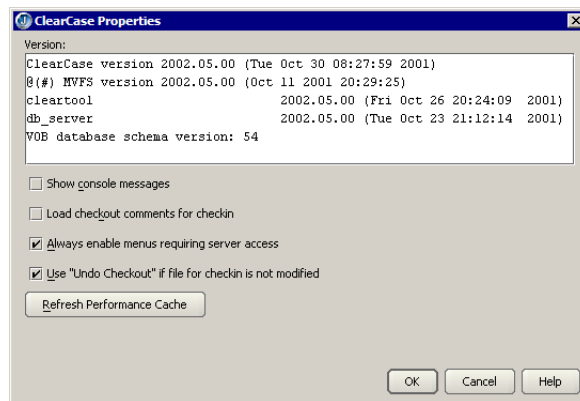
- 2 Choose ClearCase from the list of version control systems in the dialog box.
- 3 Click OK or press *Enter* to close the dialog box.

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

**Important** If the ClearCase entry in the Version Control System list is disabled, JBuilder was not able to find the necessary client executable files. Check your environment settings.

## 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 ClearCase Properties dialog box provides the following features:

- **Show Console Messages check box:** enables or disables 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.
- **Load Checkout Comments For Checkin check box:** enables or disables 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.
- **Always Enable Menus Requiring Server Access check box:** enables or disables the precondition of accessing the ClearCase server to get file state information to determine which menu commands to enable on the Team menu and the project

pane context menu. When checked (enabled), all commands are enabled, regardless of the file state. This tactic should improve performance by eliminating the network and server delay that would otherwise be involved. If you try to run a command that is inappropriate, based on a file's state, JBuilder opens a message box indicating that the file is in the wrong state to perform the command. This feature is enabled by default.

- **Refresh Performance Cache button:** flushes the ClearCase performance cache. The caching strategy dictates that information that is less transient, such as details about VOBs or directories known to reside in a dynamic or snapshot view, are kept indefinitely in memory. This performance cache reduces the number of requests to the ClearCase server for information, and therefore, improves performance. More volatile information, such as file state, is refreshed automatically whenever it might be possible the information is out of date. Click the Refresh Performance Cache button to manually flush the ClearCase performance cache, if needed. The cache flushes automatically when you close and restart JBuilder.

**Note** This dialog appears only when there are files under ClearCase control on the project source path.

## Finding information and commands quickly

---

JBuilder provides access to relevant version control information and commands from several convenient places, supporting many different working styles.

The history view in the content pane lets you examine prior revisions, look at diffs, read revision lists and comments, view checkins line by line, and handle merge conflicts if they arise.

In the project pane and the file tabs, JBuilder decorates the icons of files changed in version control. It provides textual notations and tool tips in the project pane describing each changed file's state. It checks the database state every 15 minutes by default. Customize these settings in the Tools|Preferences dialog box, on the Project tab's Decorations page.

Pertinent version control commands are available in these places:

- Choose the Team menu on the main menu bar for a complete list of commands.
- In the project pane, right-click these nodes:
  - Project node, in the Project tab
  - File node, in the Project tab
  - Any directory that's part of the project, in the File tab
- To pull a project from the database, choose File|New|Project.



# Chapter 23

## Working on an existing project in ClearCase

**ClearCase integration is a feature of JBuilder Developer and Enterprise**

JBuilder support includes creating a versioned object base (VOB), mounting a VOB, creating a View, starting a View, 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.


**Note** The ClearCase Tools command (Team|ClearCase Tools) opens a menu of commands to launch ClearCase tools, such as the ClearCase Explorer (Team|ClearCase Tools|Explorer) or the Check Out dialog box (Team|ClearCase Tools|Checkout "<filename>"). For many version control operations, such as checking in, checking out, or adding the active file, the commands on the Team menu can be used interchangeably with the corresponding native ClearCase implementation accessed with the ClearCase Tools command.

### 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,

- 1 Choose File|New|Project to open the Project panel in the object gallery.
- 2 Select Project For ClearCase  .
- 3 Double-click the icon, click OK, or press *Enter* to start the Project For ClearCase wizard.

This accesses the Project For ClearCase wizard, which 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 warns 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 depend 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 from their default locations.

- 5 Click OK to close the wizard.

JBuilder mounts the VOB, starts the View, and 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, indicating the name of the 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 check box 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 check box 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).

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 all of the files you asked it 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 check box 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 check box 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>".

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.

**Important** To convert a hijacked file to a checked out file (under ClearCase control), choose Team|ClearCase Tools|Update View to open ClearCase's Start Update dialog box. When you select the Leave Hijacked Files In Place option on the Advanced page of the dialog box, and click OK to update the view, the hijacked files will be treated as checked out files.

## 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, indicating the name of the 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.

**Note** If the project is using the Unified Change Management (UCM) process, do one of the following:

- Select an existing UCM activity from the drop-down list
- 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 it knows to check in.

Look at the list of files and make sure it matches what you intended.

A common mistake is to modify files without checking them out. This will cause a checkin to fail on that file.

- 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.

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 database 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. These checkins are not affected by other files in the list needing merging.

## Synchronizing project settings

Each JBuilder project is administered by a project file. 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. This allows you to control when and whether global settings get posted to the database.

- 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.

**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. Child project files are updated with the rest of the project.

To maintain the project file in version control

- 1 Open the Sync Project Settings submenu, available from either of these places:
  - Team|Sync Project Settings
  - Right-click the project file in the project node and choose ClearCase|Sync Project Settings.
- 2 Pull or post the project file:
  - Sync Project Settings|Pull Latest "<projectfile>.jpx".
  - Sync Project Settings|Post Current "<projectfile>.jpx".

The dialog box for pulling or posting the project file appears.

- 3 Click OK to perform the operation on the project file.
- 4 Click OK to close the dialog box.

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.

### See also

- "Creating and managing projects" in *Building Applications with JBuilder* to understand project files better.

## Changes and commands across the project

JBuilder provides tools which allow you to view the status of all the changed files in your project. Additional features let you determine which files to display in your workspace, which files to share, and which commands to run against multiple files.

**Table 23.1** ClearCase: Project-wide version control status, files, and commands

Tool	Show status	Set files	Run commands
Project pane decorations	X		
Status Browser	X	X	
Commit Browser	X	X	X
File Include Lists		X	

The Status Browser is a viewing tool. The Commit Browser incorporates the viewing features of the Status Browser, and adds context-aware ClearCase command functionality.

Access the Status Browser or Commit Browser in one of these ways:

- Choose Team|Status Browser or Team|Commit Browser.
- Right-click the project node and choose either ClearCase|Status Browser or ClearCase|Commit Browser.
- Right-click the project directory in the Files tab of the project pane and choose either ClearCase|Status Browser or ClearCase|Commit Browser.

These browsers have two pages:

- Changes/Commits page
- File Include Lists page

When you're done with the Status Browser, click OK to close it. It's purely a viewing tool, so nothing is changed.

When you're done with the Commit Browser, click the Commit button at the bottom:

- All of the commands you specified for the displayed files get executed.
- All of your comments are applied as specified.
- Files with No Action specified are left unchanged in relation to ClearCase.
- The Commit Browser closes and a progress dialog appears, displaying the progress of the version control operations.

Occasionally, updating a file creates a *merge conflict*. This happens when there's a difference between workspace and database versions in the same physical part of a file. JBuilder does not check conflicted files in. Merge conflicts must be resolved within ClearCase.

## Using the Changes and Commits pages

**In both Status  
Browser and Commit  
Browser**

These pages are very similar. The Status Browser uses the Changes page, which displays changes in files and directories. The Commit Browser uses the Commits page, which adds ClearCase commands and support for comments.

These pages consist of the following elements, starting in the upper left and going clockwise:

Element	Location	Status Browser	Commit Browser
Tree view	(Upper) left	X	X
File table	Upper right	X	X <sup>1</sup>
Summary Comment panel	Lower left		X
Tabbed source views	Lower right	X	X <sup>1</sup>

1. There is added functionality for this element in the Commit Browser.

**In both Status  
Browser and Commit  
Browser**

## Tree view

The tree view displays all of the changed files in their hierarchy. In the Commit Browser, it also provides access to commands which you can apply recursively, against every file below the level of the selected node.

To display this	Do this
All the files in the project (shown in the file table)	Select the Full List node in the directory tree
Subdirectories in the directory tree	Expand the module or parent directories
Files for an individual directory (shown in the file table)	Select the directory node in the directory tree
<b>Additionally, in the Commit Browser,</b>	
Apply ClearCase commands recursively	Right-click a node

**In both Status  
Browser and Commit  
Browser**

## Table of changed files

This table lists the changed files which belong to the node selected in the tree view on the left side of this page. Select a file in this table to view its source in the tabs below it in the dialog box.

<b>Action</b> <sup>1</sup>	Provides context-sensitive ClearCase commands.
<b>Status</b>	Displays the file's current state in relation to ClearCase.
<b>File Name</b>	Displays the name of each file that has been changed in relation to ClearCase.

1. Only the Commit Browser uses the Action column. Both the Status Browser and the Commit Browser use the Status and File Name columns.

In the Commit Browser, the Action column provides ClearCase actions in a drop-down list for each file.

**Table 23.2** ClearCase Commit Browser: Version control options

Option	Description
Add	Add this file so it's stored in the VOB.
Commit	Check in the change to the database.
Delete	This file has already been removed from the VOB; this removes it from your workspace.
Get	This file has already been added to the database; this checks it out to your workspace.
No Action	This changed file will not be touched by a ClearCase operation of any kind. It will be exactly as you left it before you invoked the Commit Browser.
Rename Back To "<orig_filename>"	This file has been renamed or moved in the workspace. When you click Commit, the file in the workspace will be renamed back to its original name or returned to its original location, and no action is performed on the file in the database.
Rename To "<new_filename>"	This file has been renamed or moved in the workspace. When you click Commit, the file in the database will be renamed to the new name or moved to the new folder, and no action is performed on the file in the workspace.
Revert	Update the workspace with the latest version of this file in the VOB, discarding all changes made since you checked the file out.

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 database or Get.

Condition	Listed status	Default option	Alternative options
File changed in the workspace	Changed In Workspace	■ Commit	<ul style="list-style-type: none"> <li>■ Undo Checkout</li> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ No Action</li> </ul>
File added to version control	Not In database	■ Add	<ul style="list-style-type: none"> <li>■ Delete</li> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ No Action</li> </ul>
File deleted from version control	Not In Workspace	<ul style="list-style-type: none"> <li>■ <i>If removed from within JBuilder: Remove</i></li> <li>■ <i>If removed from outside JBuilder: Get</i></li> </ul>	<ul style="list-style-type: none"> <li>■ Exclude In Personal List</li> <li>■ Exclude In Team List</li> <li>■ Undo Checkout</li> <li>■ No Action</li> </ul>

**Note** When you add or remove files from the Commit Browser, they're automatically committed by JBuilder. When you add or remove files individually from a menu, they need to be committed in a separate step.

If you're solely interested in the Status Browser, skip ahead to [“Tabbed source views” on page 168](#).

## Summary Comment

**In the Commit Browser**

The Summary Comment panel is where you write a comment you want to apply to more than one file. To use it

- 1 Write a comment in this panel which applies to files in the file table.
- 2 As you go through the files in the table (upper right), check the Include Summary Comment check box at the bottom of the Individual Comment page (in the tabbed view) for the files you want to apply this summary comment to.
- 3 Optionally, write individual file comments under the Individual Comment tab.

Individual comments are appended to summary comments.

When you click the Commit button at the bottom of the page, JBuilder runs your ClearCase commands on those files and applies all comments as specified.

The Include Summary Comment check box at the bottom of the Individual Comment page allows you to specify whether or not to include the summary comment for individual files. This option is on by default, or when the individual comment is blank. If you uncheck Include Summary Comment for a file, only the individual comment will be applied when the file is committed.

When a summary comment is included, it appears before the individual comment, and both are maintained as a single comment for the revision. Summary comments are entered in the Summary Comment pane in the Commits page.

In both Status  
Browser and Commit  
Browser

## Tabbed source views

Select a file from the file table above to see its source and diffs displayed here. The appropriate source views for the file's CVS status become available. For instance, if the selected file was changed in the database, the Workspace Source, database Source, and database Diff tabbed views become available.

**Table 23.3** ClearCase: Source views

Source view	Contents
Workspace Source	This file's source code from the current workspace version.
database Source	This file's source code from the current database version.
Workspace Diff	This file's most recent changes in your workspace.
database Diff	This file's most recent changes in the database.
Complete Diff	Differences between the current version of this file in the database and the current version in your workspace.

**The Commit Browser adds one more tab:**

Individual Comment	Enter your comments describing the version control action on the file selected in the file table above these tabs.
--------------------	--

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.

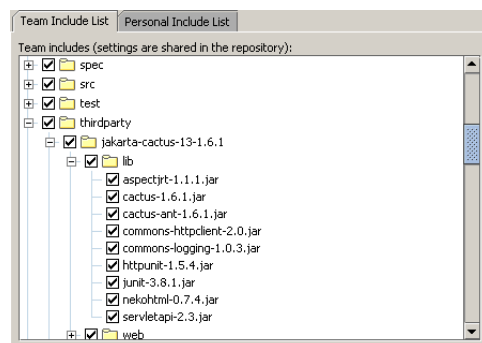
The Include Summary Comment check box is used to determine whether the summary comment should be used, either as the sole comment or else prepended to the individual comment written for this file. This is described in [“Summary Comment” on page 167](#).

## File Include Lists

In both Status  
Browser and Commit  
Browser

The File Include Lists determine which files are shared and which files are visible in your workspace. These are available from several places:

- TeamFile Include Lists
- File Include Lists page of the Status Browser
- File Include Lists page of the Commit Browser



There are two pages: Team Include List and Personal Include List.

- 1 Choose TeamFile Include Lists.
- 2 Use the Team Include List to determine which files are shared.
- 3 Use the Personal Include List to determine which files you want to keep in sight.

The files checked in the *Team Include List* are a team-wide project setting: the files that are checked here are the files that everyone needs to be able to use.

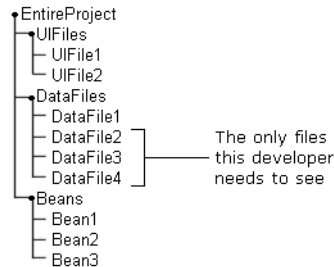
The backup files (*bak* directory) and the `<projectname>.jpx.local` file are normally excluded (unchecked). Check your company's policy about what files should be included and what should be excluded from a team project checkin.



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

The files checked in the *Personal Include List* are visible in your workspace; those that are unchecked are not, until you check them here. 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.

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



The Personal Include List page filters your view of the files, but does not affect the database or the files themselves. Files that are checked on the Team Include List but not on the Personal Include List are still available on this page, and can be rechecked on the Personal Include List at any time.

## 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, indicating the name of the file it will add.

- 2 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 operation.

- 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.

# Chapter 24

## Working on a new project in ClearCase

**ClearCase integration  
is a feature of  
JBuilder Developer  
and Enterprise**

In the 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 172](#) for more information.

To place your project into ClearCase,

- 1 Choose TeamPlace 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 work space; therefore, 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.

## Version control reference: ClearCase

**ClearCase integration  
is a feature of  
JBuilder Developer  
and Enterprise**

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

### Version control system API

---

The JBuilder IDE includes a framework for integrating any version control system (VCS), and using the VCS from within the IDE to manage Java projects. This capability is described by the VCS API, which you can use to customize an existing integration, or to design your own.

We recommend you start by reading “Introduction to PrimeTime and JBuilder OpenTools,” and “OpenTools basics” in *Developing OpenTools* to familiarize yourself with the structure and use of the OpenTools API documentation.

To learn how to integrate your own version control tool into the JBuilder IDE, or customize the existing integrations, refer to “Version control system concepts” in *Developing OpenTools* and the `com.borland.primetime.teamdev.frontend` and `com.borland.primetime.teamdev.vcs` packages.

#### See also

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

### 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).

## ClearCase resources

---

The following resources provide additional information about ClearCase.

- **Documentation:** <http://www-306.ibm.com/software/rational/support/documentation/>
- **Developer resources:** <http://www-136.ibm.com/developerworks/rational/products/clearcase/>
- **Forum:** [http://www-106.ibm.com/developerworks/forums/dw\\_forum.jsp?forum=333&cat=24&hideBody=true](http://www-106.ibm.com/developerworks/forums/dw_forum.jsp?forum=333&cat=24&hideBody=true)

P a r t   V I

# Requirements Management





# Chapter 26

## Managing requirements using CaliberRM

**This is a feature of  
JBuilder Enterprise**

CaliberRM is a collaborative, Web-based requirements management system that enables organizations to develop higher quality e-business and enterprise applications.

CaliberRM facilitates communication among project teams, providing centralized requirement data to distributed team members and allowing documented discussions about requirements and projects. Through lifecycle traceability between requirements and related development and testing tools, CaliberRM enables project teams to understand the impact of potential requirement changes on the project scope, schedule, and budget before the changes are accepted. And when requirement changes are made, CaliberRM keeps team members up to date by automatically notifying responsible individuals of the changes.

Requirements are specifications that the application or system you are building must meet. Requirements can originate from many sources such as business rules, business process models, product marketing, prototypes, development meetings and more. Requirements are stored in CaliberRM projects and are grouped according to their type.

Each requirement within a project has two numbers associated with it. One is the hierarchical number, which is determined by the requirement's placement within the project tree or hierarchy. The hierarchical number changes as requirements are added, moved or deleted. The other number is its unique ID (or serial) number. The serial number does not change, regardless of the requirement's position, and it is not reused if the requirement is deleted.

The CaliberRM plug-in allows you to view requirements, link requirements to source code files, insert requirements into source code, and determine if requirements on the server have changed. You can connect to a CaliberRM 6.0 or 6.5 server.

**Note** CaliberRM is Unicode compliant.

To use the CaliberRM plug-in, you:

- Create a connection to a CaliberRM server and login or logout
- View a CaliberRM project and baseline
- Filter requirements, so you only see the ones you want
- Drop requirements into your source code as comments
- Update comments for requirements that have been changed on the server

## Configuring a CaliberRM connection

---

To access a CaliberRM server from the plug-in, you first need to configure a connection. You can then logon to that server, select the project and baseline to use, and open the CaliberRM server. You can set up multiple connections to multiple servers and switch between them as you work.

The following steps are involved in accessing a CaliberRM server from the plug-in:

- 1 [Connecting to a CaliberRM server](#)
- 2 [Opening a CaliberRM connection](#)
- 3 [Switching between CaliberRM connections](#)
- 4 [Changing the CaliberRM project and baseline](#)

If you are having difficulty with the connection, see [“Troubleshooting the CaliberRM connection” on page 180](#) for more information.

### Connecting to a CaliberRM server

---

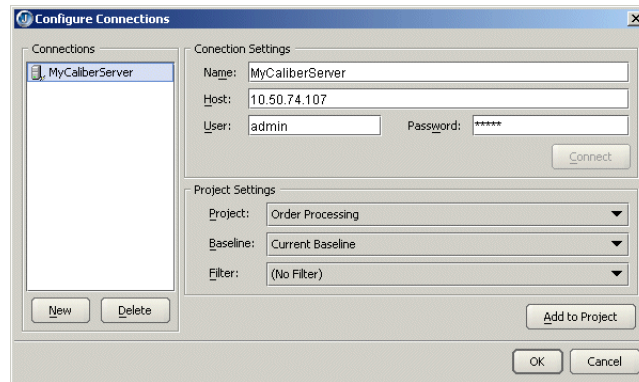
To use the CaliberRM plug-in the first time, you need to set up a connection to a server, test it, select the project and the baseline, and add the connection to your project. Once you have a single connection configured, you can add other connections, connect to those servers, and change projects and baselines.

To connect to a CaliberRM server for the first time,

- 1 Choose Tools|CaliberRM|Configure Connections to display the Configure Connections dialog box.
- 2 Click the New button at the bottom left of the dialog box.
- 3 Enter a name for the server in the Name field in the Connection Settings area of the dialog box. This name is a descriptive name only and will be displayed in the CaliberRM node in the project pane and in the message pane.
- 4 Enter the name of the CaliberRM host machine in the Host field.
- 5 Enter the user name and password in the User and Password field.
- 6 Click the Connect button to create the connection. If the connection is successful, available projects and baselines are displayed in the Project Settings area of the dialog box. If the connection is not successful, see [“Troubleshooting the CaliberRM connection” on page 180](#) for more information.
- 7 Choose the project you want to use from the Project drop-down list.
- 8 Choose the baseline you want to use from the Baseline drop-down list.
- 9 Choose the filter to apply to the requirements from the Filter drop-down list.

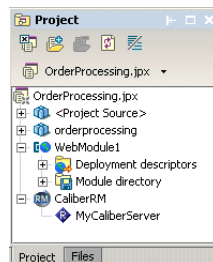
- 10 Click the Add to Project button to add the CaliberRM server to your project as a node in the project pane.



**Note** In order to access the server from the plug-in, you have to add it to your project. When you're finished, the Configure Connections dialog box will look similar to this:



- 11 You can add another connection or click OK to close the dialog box.

The server connection is created and added to your project. It is displayed as a node in the project pane. Expand the node, as illustrated below, to locate the server.



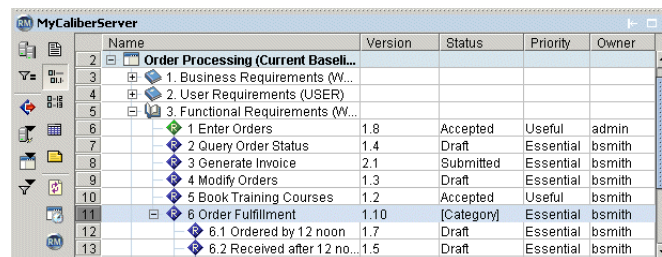
**Note** When a CaliberRM server is open, you can click the Configure Connections button  at any time to add a new connection. Use the Change Connection button  to switch to that connection. See [“Switching between CaliberRM connections” on page 180](#) for more information.

## Opening a CaliberRM connection

To open a CaliberRM connection,


- 1 Expand the CaliberRM node in the project pane.
- 2 Double-click the server you want to connect to or right-click it and choose Open.

The CaliberRM server and project are displayed in the message pane, as illustrated below.



	Name	Version	Status	Priority	Owner
2	Order Processing (Current Baseline)				
3	1. Business Requirements (W...)				
4	2. User Requirements (USER)				
5	3. Functional Requirements (W...)				
6	1 Enter Orders	1.8	Accepted	Useful	admin
7	2 Query Order Status	1.4	Draft	Essential	bsmith
8	3 Generate Invoice	2.1	Submitted	Essential	bsmith
9	4 Modify Orders	1.3	Draft	Essential	bsmith
10	5 Book Training Courses	1.2	Accepted	Useful	bsmith
11	6 Order Fulfillment	1.10	[Category]	Essential	bsmith
12	6.1 Ordered by 12 noon	1.7	Draft	Essential	bsmith
13	6.2 Received after 12 noon	1.5	Draft	Essential	bsmith

**Note** Once CaliberRM is displayed in the message pane, you can launch the full client by clicking the Launch CaliberRM button  on the toolbar. You can also right-click a requirement and choose Launch CaliberRM Viewer.

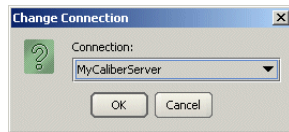
**Important** The Details panel is hidden by default. To display the panel, click the Toggle Details Panel button  on the CaliberRM toolbar.

## Switching between CaliberRM connections


If you already have a CaliberRM server open while you're updating code, you might need to switch to another server in order to view different requirements.

To switch to another CaliberRM server,

- 1 Click the Change Connection button  on the toolbar to open the Change Connection dialog box.




- 2 Choose the CaliberRM server you want to connect to from the Connection drop-down list. All configured connections are listed. Click OK to close the dialog box.

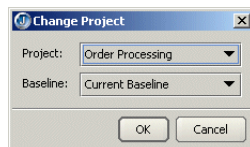
If you are not yet logged into the server you're trying to connect to, the Server Login dialog box will be displayed where you can log in. You will stay logged into the currently selected server. To log off the server, click the Logout button  on the toolbar.

## Changing the CaliberRM project and baseline

As you're working, you can change the project and/or baseline you're connected to.

To change the project and/or baseline,


- 1 Click the Change Project button  on the toolbar to display the Change Project dialog box.



- 2 Choose the project from the Project drop-down list.
- 3 Choose the baseline from the Baseline drop-down list.
- 4 Click OK to close the dialog box.

The new project and baseline are displayed in the message pane.

## Troubleshooting the CaliberRM connection

As you are working in the CaliberRM plug-in, the connection between the CaliberRM server and plug-in might be interrupted. Click the Refresh button  on the toolbar to try to restore the connection.

The following table describes the possible errors:

**Table 26.1** CaliberRM message boxes

Message box text	Description
Retrieving information from server. Press Cancel to interrupt operation.	This message is displayed when the plug-in is retrieving information from the server. The message will disappear when the server connection is established.
Failed to get requirement(s) info from server 'serverNetworkName.' Please verify the server is running and accessible and then try to refresh view.	This message is displayed when the CaliberRM server is down or a network cable was unplugged on the plug-in computer or on the CaliberRM side. Your company's network might be down. After you fix the problem, you can logout and log back in, close and re-open the CaliberRM plug-in, or click the Refresh button on the toolbar to restore the broken connection.
Failed to connect to server 'serverNetworkName.' Remote server exception.	This message is displayed if the connection is broken or unavailable when you're trying to establish a connection from the Configure Connections dialog box. Once you determine the problem, use this dialog box to establish the connection again.

# The CaliberRM UI

Once you're connected to a server and have selected the project and baseline you want to use, the CaliberRM UI is displayed in the JBuilder message pane. The toolbar is displayed on the left, the Table view is in the middle, and the Details panel is on the right.















**Note** By default, the Details panel is hidden.

## The toolbar

The toolbar on the left of the CaliberRM pane provides quick access to tasks. You use toolbar buttons to change the connection or project, show or hide the Details panel, configure the Table view, configure and choose filters, configure comments, and update comments.

The following table explains the toolbar buttons in detail.

**Table 26.2** Toolbar buttons

Button	Action	Description
	Configure Connections	Displays the Configure Connections dialog box, where you configure connections to CaliberRM servers.
	Configure Filters	Displays the Configure Filters dialog box, where you configure filters to apply to the current project. A filter determines which requirements are displayed.
	Logout	Logs off the currently selected CaliberRM server.
	Change Connection	Displays the Change Connection dialog box, where you can switch to another CaliberRM connection.
	Change Project	Displays the Change Project dialog box, where you switch the project and/or baseline you're connected to.
	Change Filter	Displays the Change Current Requirements Filter dialog box, where you select the filter to apply to the Table view.
	Toggle Details Panel	Toggles the display of the Details panel. By default, the Details panel is hidden.
	Hierarchical Numbers	Displays requirements in order by the requirement's hierarchical number, assigned by the requirement's placement within the project tree or hierarchy.
	Serial Numbers	Displays requirements in order by the requirement's serial number, its unique ID.
	Table Options	Displays the Table Options dialog box, where you configure the display of the Table view.
	Configure Comment	Displays the Configure Comment dialog box, where you configure how the requirement comment is displayed in the source code.
	Refresh	Restores a broken connection with the CaliberRM server.
	Update All Comments	Updates any comments in open source files that have changed on the server.
	Launch CaliberRM Viewer	Launches the full CaliberRM client.

## The Table view

The Table view displays CaliberRM requirements as a table. The requirement name, version number, status, priority and owner are displayed by default as illustrated below:

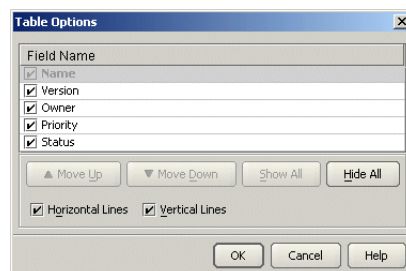
	Name	Version	Status	Priority	Owner
5	3. Functional Requirements (WHAT)				
6	Enter Orders	1.6	Accepted	Useful	admin
7	Query Order Status	1.3	Draft	Essential	bsmith
8	Generate Invoice	2.0	Submitted	Essential	bsmith
9	Modify Orders	1.3	Draft	Essential	bsmith
10	Book Training Courses	1.2	Accepted	Useful	bsmith
11	Order Fulfillment	1.10	[Category]	Essential	bsmith
12	4. Design Requirements (DSGN)				

You can apply filters to determine what requirements are displayed in the Table view. For more information, see [“Configuring filters” on page 184](#).

To configure the Table view,

- 1 Click the Table Options button  to open the Table Options dialog box.


The Table Options dialog box looks similar to this:




- 2 Click the name of the field(s) you want to display.
- 3 Select a Field Name and choose Move Up or Move Down to change where the associated column is displayed in the table. Moving a column up moves it to the left in the display; moving a column down moves it to the right. The order of the Name column cannot be changed.
- 4 Choose Show All or Hide All to show or hide all columns in the table. The Name column cannot be hidden.
- 5 Click Vertical Lines to display the table with just vertical lines. Click Horizontal Lines to display the table with just horizontal lines. Click both to display both horizontal and vertical lines.

### Displaying requirements in the Table view

You can display requirements by hierarchical number or by serial number in the Table view. The hierarchical number is determined by the requirement's placement within the project tree or hierarchy. The hierarchical number changes as requirements are added, moved or deleted on the server. The serial number is the requirement's unique ID on the server. The serial number does not change, regardless of the requirement's position, and it is not reused if the requirement is deleted.





To display requirements by hierarchical number, click the Hierarchical Numbers button  on the CaliberRM toolbar. The hierarchical number is displayed in the Name column, before the requirement name. This is the default view.

To display requirements by serial number, click the Serial Numbers button  on the CaliberRM toolbar. The serial number is displayed in the Name column, after the requirement name.


## Icons in the Table view

The Requirements icons in the Table view represent, by color, the status of the requirement.

**Table 26.3** Table view icons

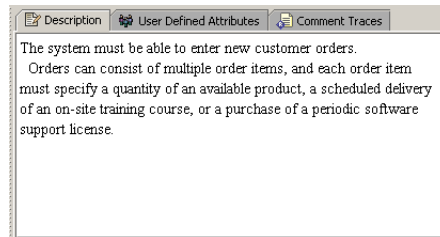
Icon	Description
 (Blue icon)	No trace has been established; no comment in code.
 (Red icon)	Trace has been established; comment is missing from source code.
 (Yellow-green icon)	Trace has been established; comment is out-of-date.
 (Green icon)	Trace has been established; comment is current.

## The Details panel

The Details panel contains three tabs: the Description tab, the User Defined Attributes tab and the Comment Traces tab. You can hide or show the Details panel with the Toggle Details Panel button  on the toolbar. By default, the Details panel is hidden.

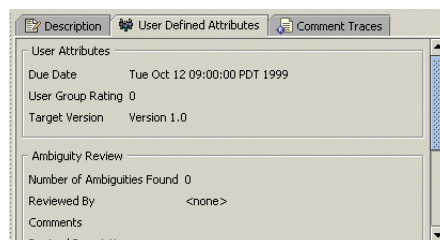
### Description tab

The Description tab contains the description associated with the requirement. This view is read-only; the description can only be changed in the full client.



### User Defined Attributes tab

The User Defined Attributes tab displays attributes associated with this requirement. This view is read-only; the attributes can only be changed in the full client.

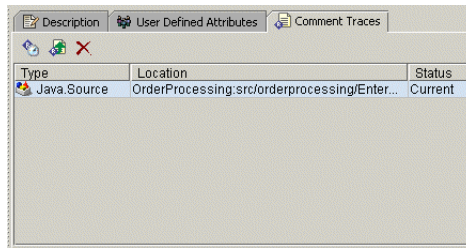


### Comment Traces tab

The Comment Traces tab displays the source file that this requirement is traced to. If the tab is empty, you can drag the requirement onto a source file to create a trace. Once a trace has been established, toolbar buttons allow you to update the comment, open the comment or delete the trace.



**Note** When you add a requirement to source code, you establish a trace to that source file.






The columns in the tab are:

- **Type** — The type of source file (Java source or HTML source).
- **Location** — The location of the source file.
- **Status** — The status of the comment. Set to Current, Missing, or Out-Of-Date.
  - **Current** — The current version of the requirement has been inserted as a comment into the source file.
  - **Missing** — The trace exists, but the source does not contain a comment for the requirement.
  - **Out-Of-Date** — The requirement on the server is different from the comment in the source file.

The toolbar buttons allow you to update comments, go to source containing the comment, or remove the trace.

**Table 26.4** Comment Traces tab toolbar buttons

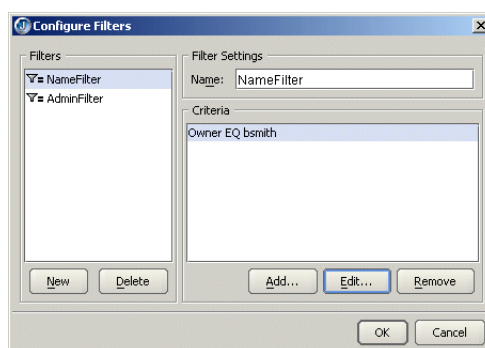
Button	Action	Description
	Update Comment	Inserts the selected requirement as a comment into the source code. Updates the selected comment if it the requirement has changed on the server.
	Open Comment	Opens the source file containing the requirement as a comment.
	Remove Trace	Removes the comment and the trace to the source file.

## Configuring filters

A filter allows you to choose which requirements to show in the Table view. For example, you can display requirements by priority or status, or display just the requirements you are responsible for. You use the Configure Filters dialog box to create and maintain filters. You can create multiple filters, however only one filter can be applied at a time.

To configure filters,

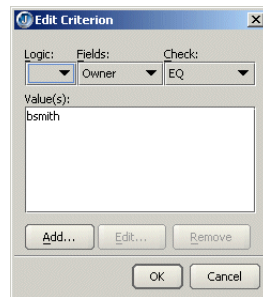
- 1 Choose Tools|CaliberRM|Configure Filters or click the Configure Filters button  on the CaliberRM toolbar. The Configure Filters dialog box is displayed.



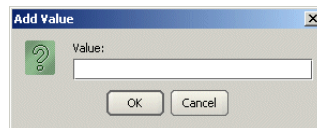
- 2 Click New in the lower left of the dialog box to create a new filter.



- 3 Enter the filter's name in the Name field.
- 4 Click Add to add filter requirements and choose the criteria to filter on. The Add/Edit Criterion dialog box is displayed.




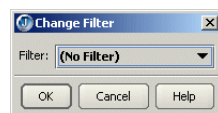
- a Choose the logic for the filter from the Logic drop-down list. Choose either no logic or the NOT operator to create an exclusion.
- b Choose the field to filter on from the Fields drop-down list. You can filter on Name, Version, Owner, Priority, Status or Description.
- c Choose the type of filter from the Check drop-down list:
  - EXISTS — Checks that the selected field and value exists.
  - EQ — Checks that the selected field equals the selected value.
  - CONTAINS — Checks that the selected field contains the selected value.
  - ONEOF — Checks that the selected field contains one of the selected value.
- d Click Add to create the value to filter on. The Add Value dialog box is displayed.



- e Add the value, then click OK when you're done.
- 5 Click OK to close the Add/Edit Criterion dialog box and create the filter.
- 6 You can add another filter or click OK to close the Configure Filters dialog box.

To select the filter to apply to the Table view,

- 1 When the CaliberRM UI is displayed, click the Change Filter button  on the CaliberRM toolbar to display the Change Filter dialog box.



- 2 Choose the filter to apply from the Filter drop-down list.
- 3 Click OK to close the dialog box.

The new requirements filter is applied. Requirements that do not meet the filter are displayed in italic text. In the following example, the requirements filter is set to display only requirements that belong to owner *bsmith*.


	Name	Version	Status	Priority	Owner
1	MyCaliberServer				
2	Order Processing (Current Baseline)				
3	1. Business Requirements (WHY)				
4	2. User Requirements (USER)				
5	Customer places order	1.9	Draft	Essential	admin
6	Customer cancels order	2.3	Draft	Essential	admin
7	Time to ship order	1.5	Pending	Essential	bsmith
8	Customer changes order	2.0	Submitted	Essential	admin
9	3. Functional Requirements (WHAT)				
10	Enter Orders	1.6	Accepted	Useful	admin
11	Query Order Status	1.3	Draft	Essential	bsmith

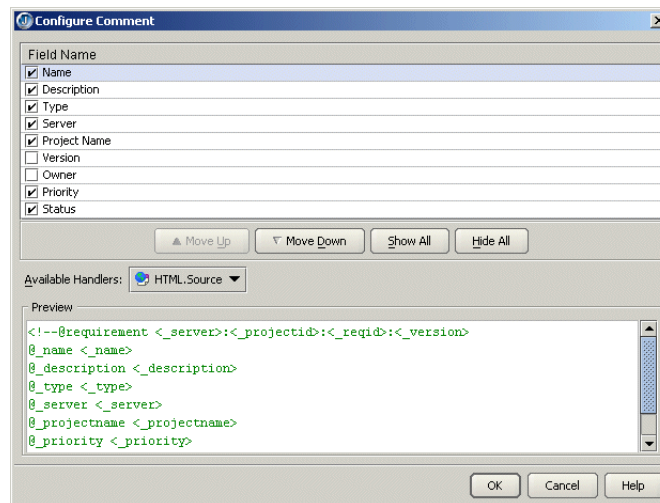
## Adding requirements to code

Requirements information can be added to source code in the current project. In Java class files, requirements are displayed as Javadoc class comments. In HTML source files, comments are displayed in HTML comment tags. Only one requirement can be added to a single source file, as the requirement establishes a trace to that file. However, the same requirement can be added to more than one source file, establishing a trace to multiple files.

Before you add a requirement as a comment or establish a trace, you can configure how the comment is displayed.

To configure the comment,


- 1 Click the Configure Comment button  on the toolbar to display the Configure Comment dialog box.



- 2 Click the field(s) you want to display in the comment tag.
- 3 Select a Field Name and choose Move Up or Move Down to change where the associated field is displayed in the comment.
- 4 Choose Show All or Hide All to show or hide all fields.
- 5 Choose HTML.Source or Java.Source from the Available Handlers drop-down list to select the format of the inserted comment. The comment preview is displayed. You cannot edit the template.

## Adding a requirement as a comment

To add a requirement as a comment,

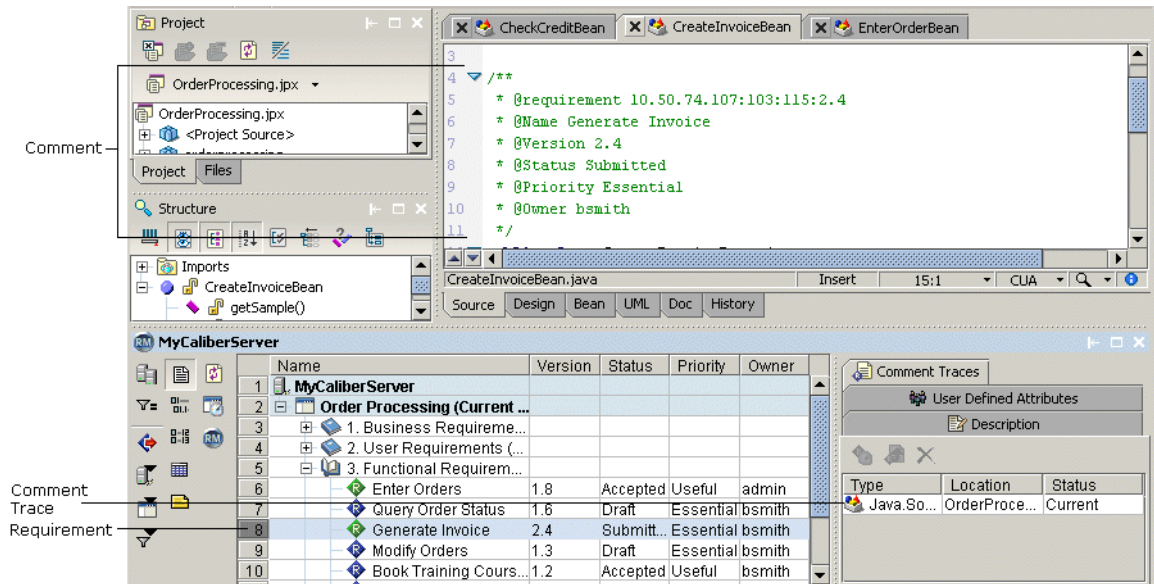
- 1 Select the requirement in the Table view.
- 2 Drag the requirement to the open source file. The comment is added to the source file as a Javadoc class comment or as an HTML comment. A trace is established in the Comment Trace tab. The status is set to Current. The Requirements icon in the Table view is set to green .

### Important


If the source files is a Java file, a Javadoc class comment is added. If the source file is an HTML file, an HTML comment is added.

JBuilder will look similar to this:

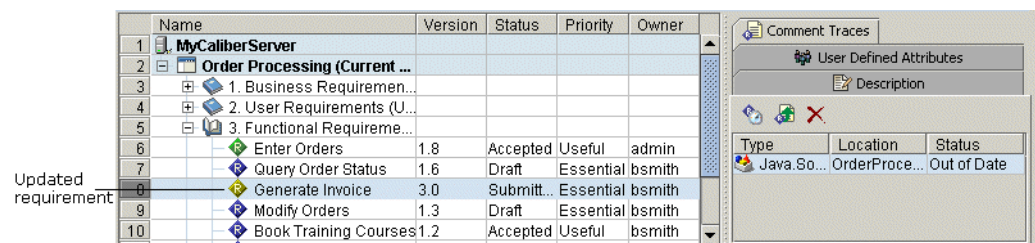
**Figure 26.1** Comment in source code



## Updating source comments



In the progress of any software project, large or small, requirements will be updated frequently. Comments in source code reflecting those requirements will become out-of-date. The CaliberRM plug-in allows you to synchronize the changed requirements with the source code comments. You can update all comments at once or update individual comments. Requirements that have been updated on the server are marked with a yellow-green Requirements icon  in the Table view.

**Figure 26.2** Updated requirement






**Note** Requirements will only be updated locally when you establish a connection to a server, for example, by connecting to a new server or by logging out and logging back in.


To update out-of-date comments for all traces in the current project,

- 1 Open a CaliberRM server connection. Requirements that have been updated on the server are marked with a yellow-green icon  in the Table view.
- 2 Click the Update All Comments button  on the CaliberRM toolbar.

All comments are updated.

To update a single out-of-date comment,

- 1 Select the updated requirement in the Table view. Requirements that have been updated on the server are marked with a yellow-green icon .
- 2 Display the Details panel, if it is not already displayed. To display the Details panel, click the Toggle Details Panel button  on the CaliberRM toolbar.
- 3 Open the Comment Traces tab in the Details panel and click the Update Comment button  on the toolbar.

**Note** To view the source file associated with the selected requirement, click the Open Comment button  on the Comment Traces tab toolbar.

# Index

## A

applying patches *See* patches

## B

Borland

- contacting 4
- developer support 4
- e-mail 5
- newsgroups 5
- online resources 4
- reporting bugs 5
- technical support 4
- World Wide Web 5

bugs, reporting 5

## C

CaliberRM 177

- adding HTML comments 186
- adding Javadoc comments 186
- adding requirements to code 186
- adding requirements to HTML source 186
- adding requirements to Java source 186
- changing baseline 180
- changing project 180
- configuring comments 186
- configuring connection to server 178
- configuring filters 184
- connecting to server 178
- displaying requirements by hierarchical numbers 182
- displaying requirements by serial numbers 182
- displaying source file trace 183
- missing comments 183
- opening server connection 179
- out-of-date comments 183, 187
- switching between connections 180
- toolbar 181
- troubleshooting connection 180
- UI 181
- updating comments 187

CaliberRM, Details panel 183

- Comment Traces tab 183
- Description tab 183
- toggling display of 183
- User Defined Attributes tab 183

CaliberRM, Table view 182

- configuring display of 182
- displaying requirements 182
- icons 183

change sets *See* patches

ClearCase integration *See* version control, ClearCase

code management *See* version control

creating patches *See* patches

CVS integration 65

*See also* version control, CVS

## D

Developer Support 4

documentation conventions 3

- platform conventions 4

## E

enabling console messages for ClearCase 156

environment variables 98

## F

File Include Lists 77, 113, 139, 168

file types

- associating 97

fonts 3

- JBuilder documentation conventions 3

## G

glossary of CVS terms 66

## J

JBuilder

- newsgroups 5
- reporting bugs 5

## M

merge conflicts

- reconciling in CVS 78
- reconciling in StarTeam 46
- reconciling in Subversion 114
- reconciling in Visual SourceSafe 132

modules

- creating in CVS 89
- creating in Subversion 119

moving files, version control  
in StarTeam 46

## N

newsgroups 5

- Borland and JBuilder 5
- public 5
- Usenet 5

## P

patches 7

- applying 7
- applying, options 8
- creating 9

performance cache, ClearCase 156

project file

- synchronizing in version control *See* version control, your VCS

project groups

- updating in CVS *See* version control, CVS

project pane

- version control notations 2, 66, 126, 157

pulling projects from StarTeam *See* version control, StarTeam

## R

reporting bugs 5

requirements management *See* CaliberRM

revision management *See* version control

## S

selecting version control systems  
    *See also* version control  
    ClearCase 155  
    CVS 89  
    StarTeam 25  
    Subversion 119  
    VSS 141  
setting environment variables 98  
SSH, using with CVS 99  
StarTeam integration *See* version control, StarTeam  
StarTeam page *See* version control, StarTeam  
Subversion integration 105  
    *See also* version control, Subversion  
synchronizing project settings 80

## T

team development *See* version control  
Team menu  
    StarTeam 15

## U

updating software *See* patches  
updating, version control  
    in StarTeam 25  
Usenet newsgroups 5

## V

version control 1  
    finding commands 2  
    finding information 2  
    how it works 2  
    resources and bookmarks 101, 151, 173  
    status information 2  
version control, ClearCase 155  
    adding new files 169  
    applying tags 170  
    checkin comments 156  
    checking in files 162  
    checking out 160  
    commands, multiple files 165  
    Commit Browser 164  
    Commit Browser, commands 166  
    Commit Browser, Commit button 165  
    Commit Browser, committing changes 167  
    creating a new VOB 171  
    File Include Lists 168  
    finding commands 157  
    finding information 157  
    menus 155  
    merging workspace and server versions 163  
    mounting a VOB/creating a project 159  
    performance 156  
    project file 164  
    Project For ClearCase wizard 160  
    showing console messages 156  
    Status Browser 164  
    status information 157  
    status of multiple files 166  
    status, multiple files 165  
    summary comments 167  
    undo checkout 161

    using on Linux 155  
    viewing diffs 168  
    viewing the connection 156  
    views, adding and editing 172  
version control, CVS 65, 69  
    adding files 81  
    branching 83  
    checking file status 86  
    checking out projects 70  
    commands, multiple files 74  
    Commit Browser 73  
    Commit Browser, commands 74  
    Commit Browser, Commit button 73  
    Commit Browser, committing changes 76  
    committing files 72  
    configuring connection 65  
    creating local repositories 91  
    CVSROOT syntax 98  
    environment variables 98  
    exporting projects 70  
    File Include Lists 77  
    finding commands 66  
    finding information 66  
    glossary 66  
    handling binary files 97  
    integration 93  
    managing file status 86  
    modules, creating 89  
    overview 65  
    project checkin 96  
    project file 80  
    pulling a project 94  
    reconciling conflicts manually 79  
    reconciling merge conflicts 78  
    removing files 82  
    secure server connection 99  
    selecting CVS 89  
    Status Browser 73  
    status information 66  
    status of multiple files 74  
    status, multiple files 74  
    summary comments 76  
    synchronizing project files 95  
    updating files 78  
    updating the project 80  
    using SSH 99  
    using watches and edits 87  
    version labelling 83  
    viewing diffs 76  
    watches and edits 87  
version control, StarTeam 13  
    adding files 39  
    administrative commands 15  
    administrative tasks 55  
    alternate property editors 52  
    checking in files 42  
    checking out a project 29  
    checking out files 44  
    commands, multiple files 31  
    Commit Browser 31  
    Commit Browser, commands 32  
    Commit Browser, Commit button 31  
    Commit Browser, committing changes 34  
    Commit Browser, StarTeam page 35  
    comparing file versions 48

- connection properties 56
- context menus 20
- File Include Lists 37
- file-level commands 16
- files, adding 39
- files, checking in 42
- files, checking out 44
- files, comparing versions 48
- files, finding 50
- files, locking and unlocking 49
- files, moving 46
- files, removing 41
- finding commands 13
- finding information 13
- folder tree pane 18
- how it integrates with JBuilder 14
- limitations of the integration 22
- locking and unlocking, files 49
- logging on and off 56
- lower pane 19
- menus 25
- placing project into database 26
- process items, adding, linking, removing 52
- process items, editing 52
- process items, setting active 52
- process items, shortcuts 59
- project file 38
- project view window 18
- project-level commands 17
- projects, managing 25
- Pull Project From StarTeam wizard 15
- reconciling conflicts manually 47
- reconciling merge conflicts 46
- removing files 41
- resources and bookmarks 61
- revision labels 57
- setting personal options 55
- shortcuts 59
- shortcuts, files 59
- shortcuts, process items 59
- StarTeam file view tab 22
- StarTeam Repository node 18
- Status Browser 31
- status information 13
- status of multiple files 32
- status, multiple files 31
- summary comments 34
- Team menu 15
- toolbar 19
- updating project file 38
- upper pane 19
- using labels 57
- viewing diffs 34
- views, creating and changing 56
- views, labels 57

- version control, Subversion 105, 107
  - adding files 116
  - checking out projects 107
  - commands, multiple files 110
  - Commit Browser 109
  - Commit Browser, commands 110
  - Commit Browser, Commit button 109
  - Commit Browser, committing changes 112
  - committing files 108
  - creating local repositories 121
  - exporting projects 107
  - File Include Lists 113
  - modules, creating 119
  - project file 115
  - reconciling merge conflicts 114
  - removing files 117
  - resolving conflicts 115
  - selecting Subversion 119
  - Status Browser 109
  - status of multiple files 110
  - status, multiple files 110
  - summary comments 112
  - updating files 114
  - updating the project 115
  - viewing diffs 112

- version control, Visual SourceSafe
  - commands, multiple files 135
  - Commit Browser 135
  - Commit Browser, commands 136
  - Commit Browser, Commit button 135
  - Commit Browser, committing changes 138
  - File Include Lists 139
  - finding commands 126
  - finding information 126
  - project file 131
  - reconciling conflicts manually 133
  - reconciling merge conflicts 132
  - Status Browser 135
  - status information 126
  - status of multiple files 136
  - status, multiple files 135
  - summary comments 138
  - viewing diffs 138

- version control, VSS 125
  - adding files 133
  - checking in files 130
  - checking out a project 127
  - checking out files 129
  - configuring project connection 125
  - creating a version label 140
  - placing project into database 141
  - programmer's guide 145
  - removing files 134
  - undoing a file checkout 130

- views in ClearCase 172

*See also* version control, ClearCase  
 Visual SourceSafe integration *See* version control, VSS  
 VSS integration *See* version control, VSS

