

SmartSVN 7.5 Manual

WANdisco plc, www.wandisco.com

2012

Contents

1	Introduction	8
2	Project Window	9
2.1	User Interface	9
2.2	Perspectives	9
2.3	Projects	10
2.4	Directory Tree and File Table	10
2.4.1	Directory States/Directory Tree	10
2.4.2	File States/File Table	10
2.4.3	State Filters	11
2.4.4	Double Click	11
2.4.5	Refresh	12
2.5	Menus	12
2.5.1	Project	12
2.5.2	Edit	13
2.5.3	View	14
2.5.4	Modify	15
2.5.5	Change Set	16
2.5.6	Tag+Branch	16
2.5.7	Query	16
2.5.8	Properties	17
2.5.9	Locks	18
2.5.10	Changes	18
2.5.11	Transactions	18
2.5.12	Window	19
2.5.13	Help	20
2.6	Changes View	20
3	Commands	27
3.1	Check Out	27
3.2	Import into Repository	28
3.3	Set Up Local Repository	29
3.4	Updating	30
3.4.1	Update	30
3.4.2	Update More	31

3.4.3	Exclude from Working Copy	31
3.4.4	Switch	31
3.4.5	Relocate	31
3.5	Local Modifications	31
3.5.1	Add	32
3.5.2	Remove	32
3.5.3	Ignore	32
3.5.4	Delete Physically	33
3.5.5	Create Directory	33
3.5.6	Rename	33
3.5.7	Move	33
3.5.8	Detect Moves	33
3.5.9	Copy	34
3.5.10	Copy From Repository	34
3.5.11	Copy To Repository	35
3.5.12	Copy Within Repository	35
3.5.13	Revert	36
3.5.14	Mark Resolved	36
3.5.15	Mark Replaced	37
3.5.16	Clean Up	38
3.5.17	Fix	38
3.5.18	Validate Working Copy Metadata	39
3.6	Commit	39
3.7	Merging	42
3.7.1	Merge	43
3.7.2	Merge from 2 Sources	44
3.7.3	Reintegrate Merge	45
3.7.4	Apply Patch	45
3.8	Properties	45
3.8.1	Edit Properties	46
3.8.2	Set or Delete Property	46
3.8.3	MIME-Type	46
3.8.4	EOL-Style	47
3.8.5	Keyword Substitution	47
3.8.6	Executable-Property	47
3.8.7	Externals	47
3.8.8	Ignore Patterns	48
3.8.9	Bugtraq-Properties	48
3.8.10	Merge Info	49
3.9	Tags and Branches	49
3.9.1	Tag-Branch-Layout	50
3.9.2	Add Tag	51
3.9.3	Tag Multiple Project Roots	52
3.9.4	Add Branch	52
3.9.5	Tag Browser	52

3.9.6	Configure Layout	52
3.10	Queries	53
3.10.1	Show Changes	53
3.10.2	Compare with HEAD	53
3.10.3	Compare with Previous	53
3.10.4	Compare with Revision	53
3.10.5	Compare 2 Files	54
3.10.6	Compare Repository Files or Directories	54
3.10.7	Log	54
3.10.8	Revision Graph	54
3.10.9	Annotate	55
3.10.10	Create Patch	55
3.10.11	Create Patch between URLs	55
3.10.12	Export Backup	56
3.10.13	Conflict Solver	56
3.11	Locks	56
3.11.1	Refresh	56
3.11.2	Lock	57
3.11.3	Unlock	57
3.11.4	Show Info	57
3.11.5	Change 'Needs Lock'	58
3.12	Remote State	58
3.12.1	Refresh Remote State	58
3.12.2	Clear Remote State	59
3.13	Change Sets	59
3.13.1	Move to Change Set	60
3.13.2	Move Up	60
3.13.3	Move Down	60
3.13.4	Delete	60
3.13.5	Edit Properties	60
3.14	Common Features	61
3.14.1	Recursive/Depth options	61
3.14.2	Revision input fields	61
3.14.3	Repository path input fields	62
3.14.4	Tag input fields	62
3.14.5	File/directory input fields	62
4	Repository Browser	63
4.1	Repository menu	63
4.2	Edit menu	64
4.3	View menu	65
4.4	Modify menu	65
4.4.1	Create Directory	65
4.4.2	Remove	65
4.4.3	Rename	65

4.4.4	Copy/Move	65
4.4.5	Edit Properties	66
4.5	Query menu	66
4.6	Window menu	66
4.7	Help menu	67
5	Transactions	68
5.1	Transactions frame	68
5.1.1	Grouping of revisions	69
5.1.2	Watched URLs	69
5.1.3	Read/Unread revisions	70
5.1.4	Display Settings	70
5.1.5	Transaction menu	70
5.1.6	Edit menu	70
5.1.7	View menu	71
5.1.8	Modify menu	71
5.1.9	Query menu	72
5.1.10	Window menu	72
5.1.11	Help menu	72
5.2	Project Transactions	72
5.2.1	Settings	73
5.3	Log Cache	73
5.3.1	Manage Log Caches	74
5.3.2	Storage	74
6	Projects	75
6.1	Managing working copies	75
6.2	Project Manager	76
6.3	Project Settings	76
6.3.1	Text File Encoding	76
6.3.2	Scan	76
6.3.3	Working Copy	77
6.3.4	Default Settings	78
7	Subwindows	79
7.1	File Compare	79
7.1.1	Comparison	79
7.1.2	File menu	80
7.1.3	Edit menu	80
7.1.4	View menu	80
7.1.5	Go To menu	81
7.1.6	Window menu	81
7.1.7	General Settings	81
7.1.8	Compare Settings	81
7.2	Properties Compare	82

7.2.1	File menu	82
7.2.2	Edit menu	82
7.2.3	Window menu	82
7.3	Compare Repository Files or Directories	82
7.3.1	Compare menu	83
7.3.2	Edit menu	83
7.3.3	View menu	83
7.3.4	Window menu	83
7.4	Conflict Solver	83
7.4.1	File menu	84
7.4.2	Edit menu	84
7.4.3	View menu	84
7.4.4	Go To menu	84
7.4.5	Window menu	84
7.5	Revision Compare	85
7.5.1	File menu	85
7.5.2	Edit menu	85
7.5.3	View menu	85
7.5.4	Go To menu	85
7.5.5	Window menu	85
7.6	Log	86
7.6.1	Log menu	86
7.6.2	Edit menu	87
7.6.3	View menu	87
7.6.4	Modify menu	87
7.6.5	Query menu	88
7.6.6	Window menu	88
7.6.7	File Export	88
7.7	Revision Graph	89
7.7.1	Revisions view	89
7.7.2	Merge Information	90
7.7.3	Graph menu	91
7.7.4	Edit menu	91
7.7.5	View menu	91
7.7.6	Modify menu	91
7.7.7	Query menu	92
7.7.8	Window menu	92
7.8	Annotate	92
7.8.1	Annotate menu	92
7.8.2	Edit menu	93
7.8.3	View menu	93
7.8.4	Revision menu	93
7.8.5	Go To menu	93
7.8.6	Window menu	93
7.9	Merge Preview	94

7.9.1	Merge menu	94
7.9.2	Edit menu	94
7.9.3	View menu	94
7.9.4	Window menu	94
8	Preferences	95
8.1	On Start-Up	95
8.2	Project	95
8.3	Authentication	95
8.3.1	Editing Profiles	96
8.3.2	Proxies	97
8.3.3	Tunnels	97
8.3.4	Passwords	98
8.4	User Interface	98
8.5	Commit	99
8.6	Conflict Solver	100
8.7	Open	100
8.8	Refresh	101
8.9	Revision Graph	101
8.10	Built-in Text Editors	101
8.11	File Compare	102
8.11.1	External Comparators	102
8.11.2	External Viewers	102
8.12	External Tools	102
8.12.1	Directory Command	103
8.13	Transactions	103
8.14	Spell Checker	104
8.15	Shell Integration (Windows)	104
8.15.1	Status Cache	105
8.16	Shell Integration (Mac OS)	105
8.17	Check for New Version	105
8.18	Customize	105
8.18.1	Toolbar (not always available)	106
8.18.2	Accelerators	106
8.18.3	Context Menus (not always available)	106
9	Shell Integration	107
9.1	Commands (Windows and OS X 10.5)	107
9.2	Commands (OS X 10.6)	107
9.3	Output Window	108
9.3.1	File menu	108
9.3.2	Edit menu	108
9.3.3	Window menu	108
9.4	Overlay Icons	108
9.5	Server Mode	109

9.6	Windows Shell Integration	109
9.7	Mac OS X Finder integration	110
9.8	Tray Icon	110
9.9	Status Cache	111
10	Plugins	113
10.1	JIRA Plugin	113
10.1.1	Workflow	113
10.1.2	Requirements	114
10.2	Trac Plugin	114
10.2.1	Workflow	114
10.2.2	Requirements	115
10.3	Remove Empty Directories	115
10.4	Quick Update	116
10.5	Plugin-API	116
10.6	Send Support Email	116
10.7	Hide Menu Items	116
10.8	Merge Info Column	116
10.9	Tag Multiple	117
10.10	Commit Message Templates	117
11	Installation and Files	118
11.1	Location of SmartSVN's settings directory	118
11.2	Notable configuration files	118
11.3	Company-wide installation	119
11.4	Command line arguments	119
11.5	JRE search order (Windows)	120
12	Advanced Settings	121
12.1	System Properties	121
12.2	Memory Limit	121

Chapter 1

Introduction

SmartSVN is a graphical Subversion (SVN) client. Its target audience are users who need to manage a number of related files in a directory structure, to control access in a multi-user environment and to track changes to the files and directories. Typical areas of application are software projects, documentation projects and website projects.

Acknowledgments

We want to thank all users who have participated in the Early Access/Beta Program of SmartSVN and in this way helped to improve it by reporting bugs and suggesting features.

Special thanks go to the SVNKit developers (<http://www.svnkit.com>) who provide the excellent Subversion base library SVNKit on which SmartSVN has been built, and to the whole SVN developer community at subversion.apache.org for making Subversion the most powerful version control system available today.

Chapter 2

Project Window

The Project Window is the central place for working with SmartSVN. In the center of the window, the main **Directories** and **Files** view show the SVN file system of your currently opened project (working copy). Various SVN commands on these directories and files are provided by the menu bar and the toolbar.

2.1 User Interface

In the bottom left area of the Project Window the **Output** view shows logged output from executed SVN commands. Depending on the command, there can be several SVN operations available for the logged files and directories.

In the bottom right the **Transactions** view (Section 5.2) collects and displays log information from the repository. The **Changes** view (Section 2.6) shows the local modifications of the currently selected file.

Only one of the aforementioned views can be “active” at any one time. Which one is displayed in the highlighted title. We will also refer to the active view as the view which “has the focus”. Menu bar actions (as well as toolbar buttons) are always referring to the currently active view.

At the very bottom of the Project Window is the status bar, displaying various kinds of information. The first and largest section of the status bar contains information on the currently selected menu item, operation progress or the repository URL of the currently selected file/directory. The second section displays information on your current selection from the **Directories** or the **Files** frame, or no information if neither of these views is active. The third section displays information on the Refresh state (see 2.4.5) of the project and the fourth section is used for progress display during the execution of SVN operations. It may either show the percentage of completion of an operation, or the total amount of bytes sent and received during the operation.

2.2 Perspectives

The layout of the Project Window can be arranged with the mouse by dragging the splitters between the various views. By dragging their titles, they can be undocked from

one position and docked to another position. You can maximize a view by double-clicking on its tab title. Double-click again on the tab title to revert it to the non-maximized state.

A complete layout configuration is called a *Perspective*. There are two perspectives available: the **Main Perspective** and the **Review Perspective**. The **Main Perspective** is primarily intended for giving you an overview of your project and repository state (Transactions). The **Review Perspective** is intended for reviewing file content changes, especially before committing them. Both perspectives can be re-configured to your needs and you may switch between them in the **Window** menu.

2.3 Projects

SmartSVN internally manages your SVN working copies in “SmartSVN projects”, as described in Section 6.

One Project Window shows one project at a time. To work with multiple projects at the same time, you can open multiple Project Windows by clicking **Window|New Project Window**. Already existing projects can be opened in a Project Window via **Open or Manage Projects**, and closed via **Close**.

2.4 Directory Tree and File Table

The directory tree and the file table show the local directories/files below the project’s root directory. `.svn` directories, *ignored* directories and files within other ignored directories are not displayed.

2.4.1 Directory States/Directory Tree

The directory tree shows the project’s directories and their SVN states, which are denoted by different icons. The primary directory states are listed in Table 2.1. Every primary state may be combined with additional states listed in Table 2.2. In case of a versioned directory, the corresponding revision number is displayed after the name of the directory. The revision will be omitted if it’s equal to its parent directory revision. If the directory hasn’t been checked out with depth Fully recursive (see 3.14.1), the check out depth will be displayed in parantheses, too. The tooltip shows detailed SVN information for the corresponding directory, similar to the contents of the file table, see below.

To *speed search* the directory tree for a certain directory, click into the tree (so the **Directories** view becomes active) and start typing the directory name. A small popup will be displayed showing the characters you have already entered. Wildcard symbols ‘*’ and ‘%’ can be used with the usual meaning.

2.4.2 File States/File Table

The file table shows the project’s files with their SVN states and various additional information. The primary file states are listed in Table 2.5 and Table 2.6. Every primary

state may be combined with additional states listed in Table 2.7. The rest of this section explains configuration options for the file table. They are only related to the current project and are also stored with the current project.

File Attributes

Tip Certain table columns require access to additional file system files when scanning the file system and therefore slow down scanning. The note within the **View|Table Columns** dialog tells you which columns these are.

Name Filters

The toolbar of the file table contains the **Filter** input field, which can be used to restrict the displayed files to a certain file name pattern. By default, simple patterns, including the wildcard symbols '*' and '%', are supported. You can also use '!' at the beginning of a pattern to invert it. For example, "!*.txt" will show all files which don't have the .txt extension.

To clear the **Filter** field, click on the button right side of the field. In the drop-down menu on the left side of the **Filter** field, you can select **Regular Expressions** instead of simple patterns. For details on the supported regular expression constructs refer to <http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html>. With **Save Pattern** you can save a pattern. Once a pattern is saved it will be displayed in the top of the drop-down menu. It can be used by selecting it and removed again by **Remove Pattern**.

Similar to the directory tree, the speed search is also available for the file table.

2.4.3 State Filters

With the menu items in the **View** menu, you can also set filters to display only files which meet certain criteria. Refer to the View menu (see 2.5.3) for details. The filter behavior can be customized in the Preferences (see 8.4) with **Hide ignored and repository-only directories according to View-menu filters** on the **User Interface** page.

2.4.4 Double Click

By default, if you double-click on a file in the file table, the file will be "opened" in one of several ways, depending on its file state:

- For an *unchanged* file which is *remotely changed* (see Section 3.12), the Compare with HEAD (see 3.10.2) command is invoked.
- An *unchanged*, *unversioned* or *added* file is opened with the file editor, see the **Query|Open** command (Section 2.5.7) for further details.
- A *conflicting* file is opened with the Conflict Solver (Section 7.4).

- All other files are opened by comparing them (Section 3.10.1).

If, for example, you want to always *open* (Section 2.5.2) the file independent of its state by double-clicking it, assign the `<Enter>`-keystroke accelerator (Section 8.18.2) to the **Query|Open** menu item.

2.4.5 Refresh

When a project is opened, the contents of the directory tree and the file table are initialized by reading the contents of the root directory into memory. Whether or not the complete project should also be read into memory at project startup can be configured in the project settings (Section 6.3).

The scanning and refreshing of the project's directories and files is generally performed in the background, so you can immediately start to work after opening a project, and you may continue to work while the project is refreshed. If a Refresh is currently in progress, the status bar shows a **Refreshing** symbol and text.

The scanning is performed *breadth-first*, so you will immediately have the complete root directory refreshed. When scanning a large working copy, you can force SmartSVN to give certain subdirectories higher priority in being scanned: As soon as possible select the (already scanned) directory in the **Directories** tree you would like to have scanned recursively. SmartSVN will then reorganize its breadth-first strategy accordingly. The same holds true for file selections: SmartSVN will give priority in scanning their common parent directory (and the path up to the root).

Tip	The initial scanning/refresh is in general much slower than subsequent refreshes due to the <i>system disk cache</i> . On Windows, you can enable the Status Cache (see 9.9) to get a first “preview” of your working copy quickly. This preview also allows most of the commands to be performed. This allows certain SVN operations to be started before the file system has been scanned.
------------	--

2.5 Menus

This section summarizes actions which are available from the various Project Window menus.

Note	You may use the Hide Menu Items plugin (see 10.7) to remove certain menu items from the menu.
-------------	---

2.5.1 Project

- **Check Out**, see Section 3.1.
- **Open Working Copy**, see Section 6.
- **Edit Working Copy**, see Section 6.

- **Remove Working Copy**, see Section 6.
- **Import into Repository**, see Section 3.2.
- **Set Up Local repository**, see Section 3.3.
- **Open or Manage Projects**, see Section 6.2.
- **Close**, see Section 6.
- **Manage Log Caches**, see Section 5.3.1.
- **Settings**, see Section 6.3.
- **Default Settings**, see Section 6.3.
- **Exit** exits SmartSVN.

2.5.2 Edit

- **Stop** stops the currently running operation. Depending on the type of operation, this action might not be applicable. On the other hand, while an operation is running, most of the other actions are not applicable.
- **Reveal in Finder** (Mac OS only) brings the *Finder* process to the front and selects the currently selected file/directory.
- **File Filter** positions the cursor in the file table's filter field.
- **Select Committable Files** selects all committable files in the file table. Because SmartSVN allows automatically adding *unversioned* or removing *missing* files for a commit, such files are also selected.
- **Select Directory** selects the deepest common directory for all selected files in the file table.
- **Select in Project** selects the currently selected files/directories from the Transactions (see 5.2) view or the **Output** area in the file table/directory tree.
- **Copy Name** copies the name of the selected file/directory to the system clipboard. If multiple files are selected, all names will be copied, each on a new line.
- **Copy Path** copies the path of the selected file/directory to the system clipboard. If multiple files are selected, all paths will be copied, each on a new line.
- **Copy Relative Path** copies the path of the selected file/directory relative to the project root directory to the system clipboard. If multiple files are selected, all paths will be copied, each on a new line.
- **Copy URL** copies the repository URL of the selected file/directory to the system clipboard. If multiple files are selected, all URLs will be copied, each on a new line.

- **Copy Revision Number** copies the revision number associated with the item in the currently active view (e.g. Directory view, Files view, Transactions view) to the system clipboard. If multiple items are selected, all their revision numbers will be copied, each on a new line. For example, if a file in the Files view is selected, this command will copy the revision number of the file to the clipboard.
- **Copy Message** copies the message of the currently selected revision in the Transactions (see 5.2) view. If multiple revisions are selected, all messages will be copied, each on a new line.
- **Clear Output** clears the output displayed in the **Output** view.
- Use **Customize** to customize accelerators, context menus and the toolbar (see Section 8.18).
- **Preferences** shows the application preferences (see Section 8).

2.5.3 View

- **Table Columns** lets you specify which file attributes will be displayed in the file table, see Table 2.3 and Table 2.4. The order of the table columns can also be defined here. (Alternatively, you can reorder them directly in the file table by dragging the column headers.) Select **Remember as default** to have the selected configuration applied to every new project.
- **Refresh**, see Section 2.4.5.
- **Files From Subdirectories** enables the recursive view showing not only files from the currently selected directory but also those from subdirectories.
- With **Unchanged Files** *unchanged* files are displayed. It is often convenient to hide them, as they are of no interest for most of the SVN commands.
- With **Unversioned Files** *unversioned* files (also within unversioned directories) are displayed.

Note	The Unversioned Files option does in no way affect the unversioned files themselves or their SVN states. Certain operations, which can work on unversioned files, will include them anyway. Parent directories of unversioned files will continue to display the <i>Direct/Indirect Local Changes</i> state. To actually ignore such files on the SVN-level you can use the Ignore command (see 3.5.3).
-------------	--

- With **Ignored Files** *ignored* files within versioned directories will be displayed. Files from ignored directories are never displayed.

- With **Files Assigned to Change Set** selected, files which have already been assigned to a Change Set (see 3.13) will be displayed. Otherwise, these files will be hidden in order to give a better overview of the files not assigned to any Change Sets. This option has no effect if the selected directory is a Change Set itself or part of a Change Set.
- With **Remote Changed Files** selected, files will be displayed which are remotely changed (see Table 3.3). Typically, this option has no effect if **Unchanged Files** is selected, because these files are shown anyway. An exception here are files which only exist remotely, i.e. files which are in *Remote* state.

2.5.4 Modify

- **Update**, see Section 3.4.1.
- **Update More**, see Section 3.4.2.
- **Exclude from Working Copy**, see Section 3.4.3.
- **Switch**, see Section 3.4.4.
- **Relocate**, see Section 3.4.5.
- **Merge**, see Section 3.7.1.
- **Merge from 2 Sources**, see Section 3.7.2.
- **Reintegrate Merge**, see Section 3.7.3.
- **Apply Patch**, see Section 3.7.4.
- **Commit**, see Section 3.6.
- **Add**, see Section 3.5.1.
- **Remove**, see Section 3.5.2.
- **Ignore**, see Section 3.5.3.
- **Delete Physically**, see Section 3.5.4.
- **Create Directory**, see Section 3.5.5.
- **Rename**, see Section 3.5.6.
- **Move**, see Section 3.5.7.
- **Detect Moves**, see Section 3.5.8.
- **Copy**, see Section 3.5.9.
- **Copy From Repository**, see Section 3.5.10.

- **Copy To Repository**, see Section 3.5.11.
- **Copy Within Repository**, see Section 3.5.12.
- **Revert**, see Section 3.5.13.
- **Mark Resolved**, see Section 3.5.14.
- **Mark Replaced**, see Section 3.5.15.
- **Clean Up**, see Section 3.5.16.
- **Fix**, see Section 3.5.17.
- **Validate Working Copy Metadata**, see Section 3.5.18.

2.5.5 Change Set

- **Move to Change Set**, see Section 3.13.1.
- **Move Up**, see Section 3.13.2.
- **Move Down**, see Section 3.13.3.
- **Delete**, see Section 3.13.4.
- **Edit Properties**, see Section 3.13.5.

2.5.6 Tag+Branch

- **Add Tag**, see Section 3.9.2.
- **Tag Multiple Project Roots**, see Section 3.9.3.
- **Add Branch**, see Section 3.9.4.
- **Tag Browser**, see Section 3.9.5.
- **Configure Layout**, see Section 3.9.6.

2.5.7 Query

- **Open** opens the selected files/directory. If the directory tree has the focus, this action will only work if a **Directory Command** has been configured in the preferences (see Section 8.12). If the file table has the focus, the file(s) will be opened in an editor. The editor to be used to open a file can be configured in the **External Tools** section of the Preferences (see Section 8.12). For files, you can specify a limit on the number of files beyond which you will be asked before the files are opened at once; for details refer to Section 8.7.

- Use **Open in Repository Browser** to open the selected directory/file in the Repository Browser (see 4).
- **Show Changes**, see Section 3.10.1.
- **Compare with HEAD**, see Section 3.10.2.
- **Compare with Previous**, see Section 3.10.3.
- **Compare with Revision**, see Section 3.10.4.
- **Compare 2 Files**, see Section 3.10.5.
- **Compare Repository Files or Directories**, see Section 3.10.6.
- **Log**, see Section 3.10.7.
- **Revision Graph**, see Section 3.10.8.
- **Annotate**, see Section 3.10.9.
- **Create Patch**, see Section 3.10.10.
- **Create Patch between URLs**, see Section 3.10.11.
- **Export Backup**, see Section 3.10.12.
- **Conflict Solver**, see Section 3.10.13.
- **Refresh Remote State**, see Section 3.12.1.
- **Clear Remote State**, see Section 3.12.2.

2.5.8 Properties

- **Edit Properties**, see Section 3.8.1.
- **Set or Delete Property**, see Section 3.8.2.
- **MIME-Type**, see Section 3.8.3.
- **EOL-Style**, see Section 3.8.4.
- **Keyword Substitution**, see Section 3.8.5.
- **Executable-Property**, see Section 3.8.6.
- **Externals**, see Section 3.8.7.
- **Ignore Patterns**, see Section 3.8.8.
- **Bugtraq-Properties**, see Section 3.8.9.
- **Merge Info**, see Section 3.8.10.

2.5.9 Locks

- **Refresh**, see Section [3.11.1](#).
- **Lock**, see Section [3.11.2](#).
- **Unlock**, see Section [3.11.3](#).
- **Show Info**, see Section [3.11.4](#).
- **Change 'Needs Lock'**, see Section [3.11.5](#).

2.5.10 Changes

- Use **Reload** to refresh the file contents from the file system and recalculate the differences.
- Use **Previous Change** to navigate to the previous change within the currently selected file. If there is no previous change, SmartSVN will select the last change of the *previous file* (as displayed in the file table).
- Use **Next Change** to navigate to the next change within the currently selected file. If there is no next change, SmartSVN will select the first change of the *next file* (as displayed in the file table).
- For **Ignore Whitespace for Line Comparison**, refer to Section [7.1.8](#).
- For **Ignore Case Change for Line Comparison**, refer to Section [7.1.8](#).
- For **Settings**, refer to Section [7.1.8](#).

2.5.11 Transactions

- **Refresh**, see Section [5.1.5](#).
- **Mark as Read**, see Section [5.1.7](#).
- **Mark All as Read**, see Section [5.1.7](#).
- Select **Show Branches and Tags** to display not only the working copy revisions but also revisions of the *trunk*, *branches* and *tags*. Refer to Section [5.1.2](#) for details.
- Select **Show Additional Watched URLs** to display not only the working copy revisions but also revisions which have explicitly been configured to be watched by **Configure Watched URLs**.
- **Ungrouped Revisions**, see Section [5.1.1](#).
- **Grouped by Days**, see Section [5.1.1](#).
- **Grouped by Weeks**, see Section [5.1.1](#).

- **Grouped by Date**, see Section 5.1.1.
- **Grouped by Authors**, see Section 5.1.1.
- **Grouped by Location**, see Section 5.1.1.
- Use **Merge** to merge the selected revision to your local working copy. If you want to configure advanced options for the merge, use the default Merge command (see 3.7.1).
- **Rollback**, see Section 7.6.4.
- **Change Commit Message**, see Section 7.6.4.
- **Configure Watched URLs**, see Section 5.1.2.
- **Settings**, see Section 5.2.1.

2.5.12 Window

- **New Project Window** opens a new Project Window for working on another project.
- **New Repository Browser** opens a new Repository Browser (see 4).
- **Show Transactions** shows the standalone Transactions Frame (see 5.1).
- **Full Screen** switches the program to full-screen mode. To get back to the normal mode, click on this menu entry again.
- **Minimize** minimizes the program window. On most platforms, to bring it back you have to click on SmartSVN in the task bar.
- **Maximize/Restore** maximizes or de-maximizes the currently active view. This action can also be performed by double-clicking on the tab title area of the respective view.
- **Hide Tool Window** hides the currently active view. This is the same as clicking on the **Close** button of the view. To bring the hidden view back, select the corresponding entry in this menu. For example, after hiding the **Directories** view, you can bring it back with **Window|Directories**.
- **Directories** puts the focus in the Directory tree (see 2.4).
- **Files** puts the focus in the File table (see 2.4).
- **Output** puts the focus in the Output view (see 2.1).
- **Changes** puts the focus in the Changes view (see 2.1).
- **Transactions** puts the focus in the Transactions view (see 5.2).

- **Main Perspective** switches to the Main Perspective (see 2.2).
- **Review Perspective** switches to the Review Perspective (see 2.2).

The subsequent content of the **Window** menu depends on which windows are currently open. For each window, there is a menu item to switch to it.

2.5.13 Help

- **Help Topics** shows the online version of SmartSVN's help.
- **Contact Support** links to our support site <http://support.smartsvn.com>.
- **Register** switches to the Professional edition.
- **Enable Connection Logging** can be used to trace and analyze problems when working with SmartSVN. The dialog will give you further instructions on how to use Connection Logging.
- Use **Obfuscate Log Cache** to remove potentially confidential information from a Log Cache so it can be sent to WANdisco PLC for debug purposes. Select the **Cache** to obfuscate, the **Output File** where the obfuscated cache should be stored and the **Map File** which contains the mapping between between real repository paths and obfuscated paths.
- **Check for New Version** connects to the SmartSVN website and checks, if there is a new version available for download. By default, this check is also performed when starting SmartSVN. You can configure the checking for new versions within the Preferences (see 8.17).
- **About SmartSVN** shows information about the SmartSVN version you're using and about your SmartSVN license. Also, the SmartSVN license agreement can be read here.

2.6 Changes View

The **Changes view** displays local changes of the file currently selected in the file table. More precisely, it displays the differences between the currently selected file, as found in the working copy, and the corresponding pristine copy of that file, as found in the repository.

Tip	The Review perspective (see 2.2) is intended to provide enough space for the Changes view, so you can switch between the Main and Preview perspective instead of resizing the Changes view.
------------	---




















Icon	State	Details
	Unchanged	Directory is under version control, not modified and equal to its revision in the repository (i.e. to its pristine copy).
	Unversioned	Directory is not under version control and hence only exists locally.
	Ignored	Directory is not under version control (exists only locally) and is marked as to be ignored.
	Modified	Directory itself has been modified compared to its revision in the repository, i.e. to its pristine copy.
	Added	Directory is scheduled for addition.
	Removed	Directory is scheduled for removal.
	Replaced	Directory has been scheduled for removal and was added again.
	Copied	Directory has been added with history.
	History-Scheduled	A parent directory has been added with history, which implicitly adds this directory with history.
	Missing	Directory is versioned, but does not exist locally.
	Added-Missing	The directory has been scheduled for addition, but is locally missing. Refer to the Fix command (see 3.5.17).
	Conflict	An update command lead to conflicting changes in directories' properties.
	Incomplete	A previous update was not completed. Run the update again to finish it.
	Root	Directory is either the project root or an external root.
	Nested Root	Directory is a nested working copy root, but not external. Refer to the Fix command (see 3.5.17).
	Obstructed	A file exists locally, but according to the pristine copy (i.e. the information from the repository) it should be a directory. Please backup the file, then remove it and update the directory from the repository.
	Phantom	The directory neither exists locally nor is versioned, but is still present in the working copy metadata (.svn directory). It's probably part of a tree conflict (see 3.5.14).
	Remote	Directory only exists in the repository. This state is only used for the remote state command (see Section 3.12).
	Unscanned	Directory has not been scanned yet (see Section 2.4.5).

Figure 2.1: Primary Directory States








Icon	State	Details
	Switched	Directory is switched (compared to its parent); see Section 3.4.4.
	Locked	Directory is locked <i>locally</i> because an operation has been interrupted before. A Cleanup (see 3.5.16) should fix the problem.
	Direct Local Changes	There are local changes to this directory itself.
	Indirect Local Changes	There are local changes to one of its files or to one of the subdirectories of this directory.
	Direct Remote Changes	There are remote changes to this directory itself, see Section 3.12.
	Indirect Remote Changes	There are remote changes to one of its files or to one of the subdirectories of this directory, see Section 3.12.
	Tree Conflict	The directory is part of a <i>tree-conflict</i> , see Section 3.5.14 for details.

Figure 2.2: Additional Directory States

SmartSVN Name	SVN info	Description
Name	(same)	File name
Revision	(same)	Current revision of the file
Local State	Schedule	Textual representation of the local state of the file
Lock	Lock Owner	Lock state of the file (see Section 3.11)
Last Rev.	Last Changed Rev.	Revision in which this file has been committed
Last Changed	Last Changed Date	Time of the last commit of the file
Text Updated	Text Last Updated	Time of the last (local) update of the file's text; this attribute is set when the content of a file has been changed by an SVN command.
Props Updated	Properties Last Updated	Time of the last (local) update of the file's properties; this attribute is set when the properties of a file have been changed by an SVN command.
Last Author	Last Changed Author	Last author, i.e. who performed the last commit on the file
Type	svn:mime-type	The file's type (see Section 3.8.3)
EOL	svn:eol-style	End-Of-Line Type of the file (see Section 3.8.4)
Keyw.	svn:keywords	Keyword substitution options of the file (see Section 3.8.5)
Needs Lock	svn:needs-lock	Whether the file should be locked before working (see Section 3.11.5)
Executable	svn:executable	Whether the file has the Executable-Property set (see Section 3.8.6)
Merge Info	svn:mergeinfo	Whether the file has the Merge Info-Property set (see Section 3.8.10): None for no Merge Info set, Empty for an empty Merge Info or Present for non-empty Merge Info. Provided by the Merge Info Column plugin (see 10.8).
Copy From	Copy From URL/Rev	Location and URL from which this file has been copied (locally). This value is only present if the file is in <i>Copied</i> state

Figure 2.3: File attributes with SVN counterparts

SmartSVN Name	Description
Remote State	Remote state of the file (see Section 3.12)
Ext.	The file's extension
Relative Directory	Parent directory of the file relative to the selected directory
File Time	The local time of the file
Attrs.	Local file attributes: <i>R</i> for read-only and <i>H</i> for hidden
Size	The local size of the file
Branch	The tag/branch to which the file is currently switched (see 3.4.4). For details, refer to Section 3.9.1.
Change Set	The Change Set (see 3.13) to which the file belongs.

Figure 2.4: File attributes without SVN counterparts
















Icon	State	Details
	Unchanged	File is under version control, not modified and equal to its revision in the repository (i.e. to its pristine copy).
	Unversioned	File is not under version control, and only exists locally.
	Ignored	File is not under version control (only exists locally) and is marked to be ignored.
	Modified (content only)	The content of the file has been modified but not its properties (compared to its revision in the repository, i.e. its pristine copy).
	Modified (properties only)	The properties of the file have been modified but not the content (compared to its revision in the repository, i.e. to its pristine copy).
	Modified (properties only)	The content and properties of the file have been modified (compared to its revision in the repository, i.e. to its pristine copy).
	Missing	File is under version control, but does not exist locally.
	Added	File is scheduled for addition.
	Removed	File is scheduled for removal.
	Replaced	File has been scheduled for removal and was added again.
	Copied	File has been added with history.
	History-Scheduled	A parent directory has been added with history, which implicitly adds this file with history.
	Remote	File does not exist locally, but only in the repository. This state is only used for the remote state (see Section 3.12).
	Conflict	An update command lead to conflicting changes either in content or in properties.
	Merged	The file has been merged. Refer to the Merge command (see 3.7.1) for details.

Figure 2.5: Common Primary File States

Icon	State	Details
	Incomplete	A previous update was not completed. Run the update again to finish it.
	Name conflict	There exists another file in the repository with the same name, only differing in upper/lower case. Such files can't be checked out on case-insensitive file systems. To fix this problem the corresponding files have to be renamed in the repository.
	Obstructed	A directory exists locally, but according to the pristine copy (i.e. the information from the repository) it should be a file. Please backup the contents of the directory, then remove it and update the file from the repository.
	Inaccessible	The file's content is not accessible, hence its state (modification) can't be determined. It's probably locked by another application.
	Phantom	The file neither exists locally nor is versioned, but is still present in the working copy metadata (.svn directory). It's probably part of a tree conflict (see 3.5.14).
	Case-Changed	The case of the file name has changed on an operating system that is case-insensitive with respect to file names. Refer to the Fix command (see 3.5.17) on how to handle such files.

Figure 2.6: Rare Primary File States

Icon	State	Details
	Switched	File is switched (compared to its parent directory); see Section 3.4.4.
	Locked (Self)	The file is locked in the repository by yourself (or more specifically, it is locked for the current working copy). See Section 3.11.
	Locked (Other)	The file is locked in the repository by some other user, see Section 3.11.
	Lock Necessary	The file needs to be locked before starting to work, see Section 3.11.5.
	Tree Conflict	The file is part of a <i>tree-conflict</i> , see Section 3.5.14 for details.

Figure 2.7: Additional File States

Chapter 3

Commands

SmartSVN provides most of the SVN command line commands in a standalone version, but also combines them into powerful higher-level commands. Common enhancements that are available for several of the following commands are explained in [Section 3.14](#).

3.1 Check Out

Use **Project|Check Out** to create a working copy from a project which is already under SVN control.

Page “Repository”

The first step is to enter the **URL** of the repository you want to check out from. On the subsequent pages you will be able to specify a subdirectory within the repository, so you don't have to append the subdirectory to the repository URL on this page.

Click **Next** to continue.

Page “Location”

After switching to this page, the repository will be scanned. A few moments later you'll see the root content of the repository. Expand the tree nodes to scan into the repository structure. For details refer to [Section 4](#).

Use **Show Revision** to define the revision of your selected directory that you want to check out. Please note that the repository contents might change when changing the revision.

Select the repository directory you want to check out and click **Next**.

When working with *trunk*, *tags* and *branches* it's not recommended to check out the whole project, because due to the increasing number of tags the working copy (not the repository) would be growing quickly, over time accumulating a lot of unneeded files on your disk. Instead you should check out only the *trunk* or a certain tag or branch and if necessary switch (see [3.4.4](#)) to another location. SmartSVN tries to detect whether you are going to check out a whole project instead of a single trunk/branch and will warn you accordingly.

Sometimes you won't need to check out the complete trunk/branch of a project, but only a certain sub-directory. Certain features (such as tags) won't work on sub-directories, hence SmartSVN will ask you whether to check out the necessary parent directory non-recursively. Such non-recursive check outs (also called "sparse checkouts") are efficient and recommended in a situation like this.

Page "Local Directory"

On this page you can select the local directory into which the working copy should be checked out. Use the options to define how the directory name should be created. The **Checkout Directory** depends on these options and always shows the final directory into which the checkout will occur (i.e. where the root `.svn-` directory will be created).

When deselecting **Check out recursively**, you will only check out the selected repository directory itself, but no subdirectories. Later you may choose to check out certain subdirectories with Update More (see 3.4.2). Non-recursive checkouts can be useful if you wish to skip certain *modules* of a project.

Click **Next** to proceed.

Page "Project"

On this page you can select whether to check out a working copy, i.e. to create the necessary `.svn/` structure, or to simply **Export** the files from the repository.

With **Check out a working copy**, SmartSVN will create a working copy for your checkout source. In this case you may select **Add a new project** for this working copy, specify the project's name and specify an optional group (Project Manager (see 6.2)) to which the project will be added. You may select **Add to current project** to add the working copy to the currently open project (if present). Or you may select **Don't manage as project** to just create a temporary project for this working copy.

With **Export only**, SmartSVN will just export the files from the repository without creating the `.svn/` subdirectory, meaning you won't be able to perform the usual SVN commands on these exported directories and files.

Click **Next** to proceed.

3.2 Import into Repository

Use **Project|Import into Repository** to add an unversioned local directory to the repository and to create the corresponding SmartSVN project. Only the specified directory will be put under version control using this command. Use the Add (see 3.5.1) and Commit (see 3.6) commands to import other files and directories of the project individually into the repository.

Page "Local Directory"

Select the unversioned **Directory** which you want to import into the repository.

Page “Repository”

Choose the **Repository** into which you want to import.

Page “Location”

After switching to this page, it takes a few moments until the first level of the repository is scanned. If you look into deeper levels of the repository by expanding the directory nodes, these levels will be scanned also. For more details refer to Section 4. Use the **Create Directory** tool button to create new directories in the repository.

Note You can create directories recursively in one go, by specifying the directories separated by /. This helps to avoid cluttering up the Log, as only one revision for creating all of these nested directories will show up.

After you’ve created the necessary directory structure in the repository, select the directory that should be linked with the root of your local project and click **Next**.

Page “Project”

On this page you can configure to which project the imported working copy will be added. You may select **Add a new project** for this working copy, specify the project’s name and specify an optional group (see Project Manager (see 6.2)) to which the project will be added. You may select **Add to current project** to add the working copy to the currently open project (if present). Or you may select **Don’t manage as project** to just create a temporary project for this working copy.

Configuring the project and doing the final import

The result of this command will be a new project, for which only the local root directory is under SVN control. This gives you many possibilities to configure which files/directories of your local file system should actually be versioned in the repository. From the **Edit** menu you can use **Add** and **Ignore** on files and directories. Furthermore, for files you can adjust properties using the respective commands from the **Properties** menu. After the project has been fully configured, use **Modify|Commit** to do the final import into the repository.

3.3 Set Up Local Repository

Use **Project|Set Up Local Repository** to set up a new local SVN repository and optionally *svnserve* to access this repository.

To use this command you need to have a local installation of the *Subversion command line binaries*. You can download them from <http://subversion.tigris.org>. It’s recommended to have these binaries and the necessary libraries on your operating system *path*. Enter the full path to **svnadmin** and **svnserve**.

Note	When proceeding with Next SmartSVN will perform some basic correctness checks on the chosen files by executing <code>svnadmin --version</code> and <code>svnserve --version</code> , respectively. Later on SmartSVN needs to be able to execute <code>svnadmin create [repository]</code> and <code>svnserve -d -r [repository-root]</code> , respectively.
-------------	---

On the **Repository** page, enter the **New Repository Location** where the repository will be created.

On the **Username** page, enter a **Username** and **Password**; the associated user which will have *write*-access to the newly created repository. Anonymous access will be restricted to *read-only*.

Note	SmartSVN will configure the file <code>conf/svnserve.conf</code> (in the selected repository directory) to use the password file <code>conf/passwd</code> . Later on you can add users and change usernames and passwords in this file.
-------------	---

After the repository has been created and configured successfully, you may choose to **Start 'svnserve' automatically when accessing the repository**. Refer to Section 8.3.1 for details. Select **Proceed with importing files into the repository** to continue with the Import into Repository wizard (see 3.2).

3.4 Updating

Updating from the repository can be done either with a simple update of the working copy or by switching the working copy to another location/revision. The following commands are available from the **Modify** menu.

3.4.1 Update

Use **Modify|Update** to get the latest changes or a specific revision from the repository for the selected files/directory.

Select **HEAD** to get the latest changes. To get a revision, select **Revision** and enter the revision number. Select **Recurse into subdirectories** to perform the update command not only for the current selected directory, but also for all subdirectories.

Advanced options

For *sparse working copies*, the Update will not pull in files/directories of repository subtrees that haven't been checked out yet. Select **Set depth to working copy** to get new subtrees as well (according to the selected **Depth** option).

When selecting **Allow unversioned obstructions**, SmartSVN will continue to update new files from the repository for which locally unversioned files already exist. Otherwise the update will be cancelled in such situations, giving you the chance to cleanup these locally unversioned files beforehand.

Use **Include Externals** to descend into externals (see 3.8.7).

3.4.2 Update More

Use **Modify|Update More** to get locally missing directories and files from the repository for a foregoing non-recursive Update or Check Out (see 3.1).

Update More checks for the currently selected directory whether there are subdirectories or files that haven't been checked out yet. They are presented in a list and you can select one or more of them to update. **Recurse into subdirectories** specifies whether the selected entries should be updated or checked out recursively.

To get rid of locally checked out directories, use the *inverse operation* Exclude from Working Copy (see 3.4.3).

3.4.3 Exclude from Working Copy

Use **Modify|Exclude from Working Copy** on one or more directories to locally exclude them from the working copy. The directories won't be removed from the repository, but will simply be ignored during subsequent Updates (see 3.4.1). To get excluded directories back, use the *inverse operation* Update More (see 3.4.2).

3.4.4 Switch

Use **Modify|Switch** to switch the selected directory or file to another repository location.

Select **Trunk** to switch back from a branch or tag to the main trunk. Select **Branch** or **Tag** and enter the branch or tag name to switch to that branch or tag. Select **Other URL** to switch to an arbitrary URL within the same repository.

You can either switch to the selected location **At HEAD** or at a specific **Revision**. Select **Recurse into subdirectories** to perform the switch command not only for the currently selected directory, but also for all subdirectories. Regarding the **Advanced** options, refer to the Update command (see 3.4.1).

3.4.5 Relocate

Use **Modify|Relocate** to change the repository for the selected directory (and subdirectories) of your local working copy. Typically, this command is used when the URL/structure of an SVN server has changed.

Relocate Directory shows the directory, relative to the project's root directory, which will be relocated. **From URL** displays the repository root URL of the selected directory, if this information is available locally. Otherwise it displays the complete repository URL of the directory. With **To URL** you can now specify the replacement string for **From URL**: Relocate will then search within the selected directory and subdirectories for URLs starting with **From URL** and replace the corresponding part by **To URL**.

3.5 Local Modifications

Local commands can be performed without a connection to the repository. They are used to prepare the working copy state for a final commit. The following local commands are

available from the **Modify** menu.

3.5.1 Add

Use **Modify|Add** to schedule files or directories for addition to SVN control.

In case of directories you have the option to **Recurse into subdirectories**, which, when selected, causes all subdirectories and files from subdirectories to be added as well.

When a file is added, SmartSVN automatically applies certain properties to the file. Most important is the automatic detection of the file's MIME-Type (see 3.8.3), which can basically be *text* or *binary*. Further property defaults can be specified in the project settings (see 6.3).

Tip Automatic detection can be overridden by the **Binary Files** project settings (see Section 6.3.3).

3.5.2 Remove

Use **Modify|Remove** to schedule the selected files/directory for removal from SVN control.

Select **Remove from SVN control and delete locally** to schedule the files/directory for removal and to also delete them locally. Select **Just remove from SVN control** to schedule for removal only. After committing the changes, the files/directories will remain as unversioned.

By default, SmartSVN refuses to remove files or directories that have local modifications, as well as directories that contain modified or unversioned files. Select **Force Removal** if you wish to perform the removal of such items anyway.

3.5.3 Ignore

Use **Modify|Ignore** to mark unversioned files or directories as to be ignored “locally”. This is useful for files or directories which should not be put under SVN control. These are usually temporary, intermediate or automatically generated files, like C's `.obj` or Java's `.class` files, or directories containing such files.

Local ignore patterns are stored within the working copy (in the `svn:ignore` property of the corresponding parent directories) and will be committed. Therefore, to have a file locally ignored, its parent directory must either be ignored as well, or be versioned, so that the necessary `svn:ignore` property can be stored there. Hence, when trying to ignore a file or directory within another unversioned directory, SmartSVN will ask you to add this parent directory. In addition to *local ignore patterns*, you can configure *global ignored patterns* in the project settings (see 6.3).

You can select **Ignore Explicitly** to add each selected file/directory explicitly to the ignore list. If SmartSVN detects a common pattern for the selected files/directory, it will also allow you to **Ignore As Pattern**.

This command is a shortcut for editing the `svn:ignore` property directly by **Properties|Ignore Patterns**. Refer to Section 3.8.8 for details.

3.5.4 Delete Physically

Use **Modify|Delete Physically** to delete local files, or unversioned or ignored directories.

Warning! Be careful before deleting a file (or directory) as there will be no way to recover unversioned items.

3.5.5 Create Directory

Use **Modify|Create Directory** to locally create a directory within the currently selected directory.

Enter the **Path** of the subdirectory that will be created. The path may consist of multiple directory names, separated by “\” or “/” to create multiple directories at once. Select **Schedule for addition** to schedule the created directory/directories for addition to SVN control, see Section 3.5.1.

3.5.6 Rename

Use **Modify|Rename** to rename a file or directory which is already under SVN control. The file with the old name will be scheduled for removal, the file with the new name for addition. This command will preserve the file’s history.

3.5.7 Move

Use **Modify|Move** to move and/or rename a file or directory which is already under SVN control. The file with the old name will be scheduled for removal, the file with the new name for addition. This command will preserve the history of the moved item.

There is also a special mode of this command that can be used to tell SmartSVN “after the fact” that a file was moved. For this to work, only two files can be selected: One that is missing or removed, and another that is unversioned, added or replaced. SmartSVN will then remove the missing file (if necessary), add the unversioned file (if necessary), and connect the history of the added file to that of the removed file.

Tip You can also use Drag-And-Drop to copy or move files and directories.

3.5.8 Detect Moves

Use **Modify|Detect Moves** to convert already performed “manual” moves (including re-names) of files to “SVN” moves. Typically, you will not perform moves within SmartSVN itself, but with system commands, through an IDE, etc. One such external move results in a missing file and a new unversioned file. Both files could then be changed and committed. This will result in the repository content being up to date, but will not preserve the relationship between both files (which is actually one moved file). This especially affects the log of the added file: It will start at the committed revision and won’t include the revisions of the removed file. To preserve the relation (and hence history/log), a “post-move”

on both files has to be performed. **Detect Moves** can detect such already performed “manual” moves based on the file content and displays the corresponding suggestions of which files could be “post-moved”.

Invoke **Detect Moves** on a set of missing and unversioned files for which “post-move” should be detected. Depending on the number of selected files, the operation might take a while. The results will be displayed in terms of a list of possible “post-moved” files pairs.

Suggestion displays the detected move in a descriptive manner. If you agree that the corresponding file pair actually represents a move that has happened, keep it selected so the corresponding “post-move” will be performed. **Similarity** can be helpful for this decision. It is entirely based on the comparison of the file contents and denotes the calculated likelihood for the file pair to be an actual move.

For more details, **Target** displays the name of the unversioned (i.e. new) file. **Source** displays the name of the missing (i.e. old) file. If the name of the file has not changed, i.e. **Target** would be equal to **Source**, **Source** is omitted. In the same manner **Target Path** displays the path of the new file and **Source Path** displays the path of the old file. Again, **Source Path** will be omitted if it is equal to **Target Path**.

There can also be more than one possible **Source** for a specific **Target**. In this case SmartSVN always suggests the best matching **Source**, i.e. the file for which the highest **Similarity** value was calculated, and **Alternatives** shows the number of possible alternative sources. Use **Compare** to compare the currently selected **Source** and **Target** file with the File Compare (see 7.1). Use **Alternatives** to select an alternative source to be used instead of the original suggestion. Finally, if you consider a particular suggestion and all available **Alternatives** incorrect, you may deselect the suggestion so that no “post-move” will be performed for the respective **target**.

Click **OK** to perform the selected “post-moves”.

3.5.9 Copy

Use **Modify|Copy** to create a copy of a file or directory which is already under SVN control. This command will preserve the history of the copied item.

Select the **Target Directory** under which the copy of the file/directory will be created, and specify the **New Name**.

There is also a special mode of this command that can be used to tell SmartSVN “after the fact” that a file was copied. For this to work, exactly two files must be selected: One that is versioned, but not added or replaced, and another that is unversioned, added or replaced. SmartSVN will then add the unversioned file (if necessary) and connect the history of the added file to that of the other file.

Tip	You can also use Drag-And-Drop to copy or move files and directories.
------------	---

3.5.10 Copy From Repository

With **Modify|Copy From Repository** you can copy a file or directory from the repository to your local working copy. This command can be used for recovering deleted files and directories from earlier revisions.

Repository is the repository of your local working copy, it can't be changed as copies can only be performed within the same repository. For **Copy** enter the file/directory to be copied, along with its **Source Revision**. Specify the local directory **Into Local** into which the file/directory should be copied. **With Name** will be the name (i.e., last component of the path) of the restored file/directory.

3.5.11 Copy To Repository

With **Modify|Copy To Repository** you can copy the selected local file/directory to the repository. This operation can be used to create tags, although SmartSVN provides more convenient commands for this task (see Section 3.9).

Repository is the repository of your local working copy, it can't be changed as copies can only be performed within the same repository. The local file/directory **Copy Local** will be copied to the project's **Repository**. The target directory is **Into Directory**. **With Name** will be the name (i.e. last component of the path) of the resulting file/directory. Because the copy is directly performed into the repository, you have to specify a **Commit Message**.

Use **Externals Revisions** to specify how to handle external revisions (see 3.8.7). This option is only relevant for externals which have their revisions set to **HEAD**. By default, **Leave as is** will not modify any external revisions. Choose **Fix all** to have all revisions set to their current values, as present in the working copy. Choose **Fix except below** to have all revisions set to their current values except externals pointing to the specified location, or some subdirectory of this location.

Only when fixing externals you can make sure that later checkouts of the copied location will produce exactly the same working copy. Otherwise, externals which have been left at **HEAD** will continue to bring the latest revisions of that external, which are in general not equal to it at the time of creating the copy.

3.5.12 Copy Within Repository

With **Modify|Copy Within Repository** you can perform copy operations that take place entirely within a repository. This is for instance a convenient and fast way to create repository tags/branches.

Select the **Repository** within which the copy should be performed. **Copy From** and the **Source Revision** specify the copy source. For **Copy** you can either select to copy **To** or to copy **Contents Into**. In the case of copy **To**, the source will be copied into the **Directory** with its name set to **With Name** (last component of the path). For copy **Contents Into**, the contents (files and directories) of the source will be copied directly into the **Directory** with their corresponding names. Because the copy is directly performed in the repository, you have to specify a **Commit Message**.

Note	This copy operation is in fact not a local operation, as it requires no working copy. Due to its close relationship with the other copy operations we have nevertheless put it into the chapter "Local Modifications".
-------------	--

3.5.13 Revert

Use **Modify|Revert** to revert the local changes of the selected files/directories. In case of directories you have the option to **Revert Recursively** (i.e. to recurse into subdirectories). If deselected, only the properties of the directory itself will be reverted.

- Added and copied files/directories will be unscheduled for addition and returned to unversioned state.
- Removed files/directories will be unscheduled for removal and restored with their content and properties taken from the pristine copy.
- Replaced files/directories will be unscheduled for replacement and restored with their content and properties taken from the pristine copy.
- Modified files/directories will be restored with their content and properties taken from the pristine copy (overwriting local changes!).
- Missing files will be restored with their content and properties taken from the pristine copy. Missing directories can't be restored, because the pristine copy is also missing. You have to freshly Update (see 3.4.1) them from the repository.
- Conflicted files/directories will be restored with their content and properties taken from the pristine copy (ignoring local changes which caused the conflict!).
- For Case-changed files their original file names will be restored and modifications in contents/properties will be reverted.

Warning! Be careful before reverting a file or directory as all local modifications will be lost.

3.5.14 Mark Resolved

Use **Modify|Mark Resolved** to mark conflicting files (see Table 2.5) or conflicting directories (see Table 2.1) as resolved. You have to resolve conflicts to be able to commit the files/directories.

In case of directories you have the option to **Resolve files and subdirectories recursively**. If selected, all conflicting files and directories within the selected directory will be resolved. Otherwise only the property conflicts of the directory itself will be resolved.

Regarding the **File Content**, use **Leave as is** to apply no further modifications to resolved files. Use **Take working copy** to replace the contents of resolved files with their contents as they were before the update/merge. Use **Take new** to replace the contents of resolved files with the contents of their corresponding pristine copies as they are now after the update/merge. Use **Take old** to replace the contents of resolved files with the contents of their corresponding pristine copies as they were before the update/merge.

Tree Conflicts

Certain kinds of conflicts are not directly related to the content or properties of a file (or directory) but to conflicting actions on a file/directory. Such conflicts are called *tree-conflicts*.

Tree conflicts are similar to *normal* conflicts as conflicting files/directories can't be committed before they have been resolved. The Local State (see 2.4.2) column for files shows details for a tree conflict, if present. File and directory tooltips display this information as well.

Example

You have modified file `foo.txt` in your working copy. Your co-worker has renamed `foo.txt` to `bar.txt` and has committed this change. When updating from the repository, you will receive `bar.txt` but because of your local modifications to `foo.txt` this file will not be deleted, but re-scheduled as copied from itself (but the revision before the update). Furthermore, `bar.txt` will receive your local modifications of `foo.txt`. This represents a tree conflict. There are different kinds of tree-conflicts, for a detailed analysis refer to: <http://svn.collab.net/repos/svn/branches/1.6.x/notes/tree-conflicts/>

3.5.15 Mark Replaced

Use **Modify|Mark Replaced** to mark modified files or a directory as *replaced*, see Table 2.5 for details.

Marking modified files or a directory as *replaced* does not affect the contents of the files or directories, but only the meaning of the commit and the history of the directory/files. This can be useful to show that the content of a directory/files is not related to its previous revision. The Log (see 3.10.7) of such a directory/files will not go beyond the replacement revision, meaning that the directory/files has been created at that revision.

Example

For example, we have a Java interface `Person.java` and one implementing class `PersonImpl.java`. As the result of a refactoring, we are getting rid of the interface `Person.java` and renaming the class `PersonImpl.java` to `Person.java`. This results in a *removed* file `PersonImpl.java` and a *modified* file `Person.java`.

When simply committing these changes, this would mean that the class `PersonImpl.java` has been removed and the interface `Person.java` has been changed to a class `Person.java`, with no history except that of the interface.

Taking a closer look at this situation, it would be better to do a commit meaning that the interface `Person.java` has been removed and the class `PersonImpl.java` has been renamed to `Person.java`. At least that was the intention of our refactoring and it would also mean to preserve the history of `PersonImpl.java` for `Person.java`.

To achieve this, we will use **Mark Replaced** on `Person.java` and then we will use **Move** on `Person.java` and `PersonImpl.java`, performing a “post-move” between both files (for details refer to Section 3.5.7), yielding a *removed* `PersonImpl.java` and a *replaced* `Person.java`, which has its history (**Copy From**) set to `PersonImpl.java`.

3.5.16 Clean Up

Use **Modify|Clean Up** to clean up unfinished SVN operations for the selected directory (and all subdirectories). Cleaning up a working copy is necessary when the working copy becomes “internally” locked (in contrast to file locks, see Section 3.11). A working copy can become locked when certain SVN operations (like commit or update) are aborted. In general, cleaning up a working copy is a safe process.

Note A clean up may fail for the same reasons for which the preceding SVN operation has failed. This typically happens if certain files or directories can't be read or written. In such cases, please check whether other running processes might lock the file and whether file permissions have been set adequately.

3.5.17 Fix

Use **Modify|Fix** to fix (or “repair”) the selected directory/files. This option is only applicable for certain, unusual working copy states:

Case-changed files

SVN repositories and working copies are in general case-sensitive. This can cause problems when working on a case-insensitive operating system, like Microsoft Windows or certain file systems on Apple Mac OS, and changing the file name's case (upper-case to lower-case, etc.). Because of SVN's working copy format and the *pristine* copies, it's technically not possible to handle such a file name case change.

One solution is to avoid this situation by either only performing file name case changes on an operating system which supports case-sensitive file names, or directly in the repository by using the Repository Browser (see 4).

At any rate, a file name case change may happen on a case-insensitive operating system, e.g. because of defect software tools, etc. SmartSVN detects such invalid changes and puts the file into a *case-changed* file state, see Table 2.6. **Fix** will now change the file name case back to its original form as found within the pristine copy, resolving this problem.

Nested Roots

A *nested root* (see Table 2.1) is a working copy within another working copy which is not related to this parent working copy. SVN commands ignore such nested roots; they are simply treated as *unversioned* directories.

Nested roots typically result from moving a directory from one location to another without using the proper SVN commands, like Move (see 3.5.7). This leaves a *missing* directory at its original location and introduces a *nested root* at its current location.

Fix offers the following solutions for *nested roots*, depending on their origin:

- **Mark as Copied** will convert the nested root to a *copied* directory, with its copy location being the original repository location. This option is only available if the current location is part of the same repository as the original location.
- **Convert to Unversioned** strips off the working copy metadata (.svn directories) for this directory and all of its children. This will make the directory *unversioned*, so it can be added and committed afterwards. This option is always available but in general should only be used if **Mark as Copied** is not available, as *unversioned* directories can be added afterwards, but their history will be lost.

Added-missing directories

If a directory has been scheduled for addition (see Section 3.5.1) and has been locally deleted afterwards (i.e. the directory and its .svn subdirectory are missing), the directory is in *Added-missing* state (see Table 2.1).

Such directories are actually only a leftover entry in the parent directory's metadata directory (.svn). The resolution is to Revert (see 3.5.13) them, which is what this command will do.

3.5.18 Validate Working Copy Metadata

Use **Modify|Validate Working Copy Metadata** to check the working copy metadata (.svn directory) for possible inconsistencies, and to do some other house-keeping tasks.

3.6 Commit

Use **Modify|Commit** to commit (i.e. send) the changes in the selected files/directory to the repository.

The **Commit** wizard guides you through the commit process, starting with the “Configuration” page. Based on the choices on the “Configuration” page, the working copy will be scanned for changes, which is especially important when performing the **Commit** on a directory. Subsequent pages allow further “customization” of the commit depending on the changed files and directories found during the scanning phase.

Before the commit wizard opens, it checks whether you might have forgotten to select some files or directories, and in that case shows a warning. For details, refer to the Preferences (see 8.5). Also, a warning will be issued if you are going to commit switched (see 3.4.4) files or directories. Unless this is intended, you should switch back the corresponding entries and re-run the commit.

Page “Configuration”

This page is only present when committing one or more directories.

Select **Recurse into subdirectories** to scan not only changes from the selected local directory, but also from subdirectories.

When recursing into subdirectories, select **Descend into externals** to also scan for changes in external working copies (see 3.8.7).

When clicking **Next** the file system of your project will be scanned. This may take some time.

Page “Repositories”

This page is only displayed if the option **Descend into externals** on the **Configuration** page has been selected and at least one committable entry within an external working copy has been found. For details on externals, refer to Section 3.8.7.

Every such external working copy is listed with its **Local Path** and its **URL**. The project’s working copy itself is also listed with local path “.”. Every working copy can be individually selected or deselected for the commit by toggling the respective checkbox in the first table column (either with the mouse or with `<Space>`-keystroke).

Working copies pointing to the same repository (the **URL** identifies these) can be committed together, hence SmartSVN will have to perform as many commits as different repositories are involved in the overall commit process.

Warning! When committing to multiple repositories, every commit will create its own revision in the corresponding repository. Hence, atomicity of such commits cannot be guaranteed. This for example means that the commit to one repository can succeed while the other one fails. While fixing the failing commit another person might already have updated his or her working copy and only have received the successfully committed revision. This might result in (temporary) inconsistencies in his/her overall project.

You can choose whether to commit the selected working copies with **One commit message** or with **Individual commit messages**. If you are committing multiple working copies with different Bugtraq-Property (see 3.8.9) configurations, it’s required to use **Individual commit messages** to have the Bugtraq-Property functionality available on the **Files** page.

Page “Detect Moves”

This page is only displayed if the option **Detect moved and renamed files** in the Preferences (see 8.5) has been selected and at least one moved or renamed file pair was detected. Refer to Section 3.5.8 for details. With **Differences** you can toggle the integrated compare view. This will show the differences for the currently selected file in the lower part of the **Commit** dialog. The change display behaves similarly to the Changes view (see 2.6).

Page “Files”

This pages shows a list of all files and directories which were found to be committable according to the selected options from the **Configuration** page, and from the configuration

in the Preferences (see 8.5). To exclude a file/directory from the commit, deselect the corresponding checkbox (either with the mouse or by pressing `<Space>`-keystroke).

Note SmartSVN also displays certain kinds of files which are not committable (e.g. conflicted files, refer to Table 2.5). This is a caution not to forget to resolve these files' problems and commit them as well (if necessary).

You may review your changes by expanding the dialog with the **Differences** button. This will show the differences for the currently selected file in the lower part of the **Commit** dialog. The change display behaves similarly to the Changes view (see 2.6). Alternatively, you can also double-click a file to open a File Compare (see 7.1) frame.

For the **Commit Message** you can either enter a new message or select an older message from the message popup on the top right of the text field. The popup menu will show recently entered commit messages, and you can clear this message history with **Clear Message History** or use **Get from Log** to fetch an older commit message from the log. With `<Ctrl>+<Space>`-keystroke you can trigger a file name completion, based on all of the files that have been selected for the commit.

Depending on how Bugtraq-Properties (see 3.8.9) are configured for the current working copy, there may be an additional "issue ID" input field. The name of this field can vary, depending on the Bugtraq-Properties. Its content will be appended/prepended to entered commit message afterwards, forming the final commit message.

If the spell check (see 8.14) has been configured, SmartSVN will check the entered **Commit Message** for basic spelling mistakes. The spell check ignores file paths, i.e. strings containing a `/`, and "issue IDs" which are part of the commit message and which can be recognized by the Bugtraq-Property. For details regarding the spell check's popup menu, refer to Section 8.14.

Tip Commit messages will be displayed in various kinds of logs. Hence, a descriptive commit message will help you and your team later on if you need to go through your past changes for some reason, e.g. tracking down a bug.

By default, SmartSVN will warn you in case of an empty commit message. You can disable this warning in the Preferences (see 8.5).

Tip You may configure a *template message* using the `tsvn:logtemplate` property which has to be set on the project root. For details refer to Section 10.10.

If **Descend into externals** has been selected and multiple working copies on the **Repositories** page have been selected for committing, there will also be a topmost **Path** drop-down list. All other items on this page will be related to the selected path. In particular it will be necessary to enter a **Commit Message** for each path.

Page “Locks”

This page will only be displayed if the files/directories selected for the commit have been found to contain locked files during the scanning.

Select **Keep locks for committed files** to keep the files locked even after having them committed. Select **Unlock committed files** to unlock them after the commit. In the latter case you can additionally select unchanged but locked files which had been detected during the scan and which should be unlocked upon a successful commit as well.

Tip	You can configure whether Keep locks for committed files or Unlock committed files should be selected by default in the Project Settings (see 6.3.3).
------------	--

Click **Finish** to finally start the commit.

3.7 Merging

Merging is used to incorporate changes from one “development line” into another.

Note	Subversion’s merging has been significantly improved with version 1.5 and its “merge tracking” support. Most merging features require a Subversion 1.5 server to work. Subsequent explanations assume that you are performing the commands against a Subversion 1.5 server.
-------------	---

Two very common use cases of merging are *release branches* and *feature branches*:

- A *release branch* is typically forked off from the main development line (*trunk*) after the “release” of a new version (of the software project, of the website, etc). With the “release” the corresponding version typically goes into “production use” and gains stability while development continues on the *trunk*. Therefore a release branch will only receive problem fixes (bug fixes) from the *trunk* by merging them to the branch.
- A *feature branch* is a line of development that is being worked on in parallel to the *trunk*, for the purpose of developing a new “feature” which is brought back to the *trunk* after completion. A *feature branch* is frequently merged from the *trunk* to stay up to date with the *trunk* changes, and once the implementation of the “feature” has been finished, all relevant changes are merged back to the *trunk*.

For more in-depth information on these use cases, for examples and for general information, refer to <http://svnbook.red-bean.com/>.

Warning! As merging can often turn out to be rather tricky, here are a few recommendations that may help:

- Only do recursive merges and try to always merge on the same “merge root”, preferably the *trunk* itself or the root of a branch.
- Avoid merging into a working copy which contains mixed revisions. Therefore do an Update (see 3.4.1), preferably to the **HEAD** revision, beforehand.
- Avoid merging into working copies that are non-recursive or not completely checked out. To do so, run an Update More (see 3.4.2) on your merge root, selecting all files and directories and the **Recurse into subdirectories** option.

3.7.1 Merge

Use **Modify|Merge** to merge changes from another source branch to the selected file/directory.

Select **Trunk** to merge from the main trunk. Select **Branch** or **Tag** and enter the branch or tag name to merge changes from a branch or tag. Select **Other Location** to merge from an arbitrary location, specifying the corresponding repository and path.

Alternatively, you may select a merge source from the **History** button. It shows a list of previous merge sources you have used, as well as merge sources extracted from the `svn:mergeinfo` (see 3.8.10) property of your merge target.

Use **All Revisions** to merge all the revisions which have not yet been merged from the selected location. SmartSVN will detect them based on the present *merge tracking* information.

Example

You will typically use this option when working with a *feature branch* to keep it in sync with the *trunk*.

Warning! **All Revisions** does not work with *pre-1.5* servers (e.g. 1.4 servers).

Use **Revision Range** to manually specify multiple (ranges of) revisions to be merged from the selected location. SmartSVN will detect whether certain revisions of the specified ranges have already been merged and avoid repeating the merge. Single revisions are just specified by their revision number while ranges starting at **start** (inclusive) and ending at **end** (inclusive) are specified by **start-end**. Multiple revisions (ranges) can be specified with a separating colon (:). Certain revisions may be excluded by prefixing them with an exclamation mark (!).

Instead of entering the revisions manually, you can choose them from the revision browser (see 3.14.2). The revision browser will indicate with a green arrow which revisions have not been merged (“merge candidates”). From the **Options** button you can select **Show only mergable revision** to restrict the revision list to those merge candidates. By default, **Show all revisions** will also include revisions which have already been merged.

Example

You will typically use this option when working with a *release branch* to merge only bugfix revisions from the *trunk* to this branch.

Select **Reverse merge** to reverse the changes between the selected revisions. Internally, this is achieved by swapping the start and end revisions.

Advanced options

By default, merging takes the ancestry into account, meaning that merging does not simply calculate (and merge) the difference between two files which have the same path, but also checks if both files are actually related. For the typical merging use cases, this behavior leads to the expected results and it is also required for the merge tracking to work. You can switch this behavior off by selecting **Ignore ancestry**, however this option is not recommended unless you have a good reason to use it.

Regarding **Ignore changes in EOL-style** and **For Whitespaces** handling, refer to Create Patch (see 3.10.10).

Deselect **Recurse into subdirectories** to merge only changes to the selected directory/file itself but not it's contained files, etc. In general it's recommended to keep **Recurse into subdirectories** selected.

With **Record only** no files will be touched during the merge, and only the Mergeinfo (see 3.8.10), will be adjusted accordingly, so that the core merge tracking mechanisms consider the revisions as merged. This option can be useful to “block” certain revisions from being actually merged.

By default merging will stop when it's required to delete locally modified files, because they have been removed in the merge source. You can switch off this safety check by selecting **Force deletion of locally modified files, if necessary**.

Close the dialog with **Merge** to immediately perform the merge to the selected directory/file of the current working copy. Alternatively you may choose to **Preview** the changes that would result from the merge. Refer to the Merge Preview (see 7.9) for details.

Tip You can choose to keep the auxiliary merge files even for non-conflicting files in the Project Settings (see 6.3.3).

3.7.2 Merge from 2 Sources

Use **Modify|Merge from 2 Sources** to merge changes between two different merge sources (URLs) to the selected file/directory.

Changes are merged from one **Repository** between **From** and **To** to the local **Destination**. The last 10 merge sources will be stored and can be set using the drop-down button beside the **Repository** drop-down list. For details regarding the **Advanced** options, refer to Section 3.7.1.

Note Most merging use cases are covered by Merge (see 3.7.1) and Reintegrate Merge (see 3.7.3) and if possible these commands should be used instead.

3.7.3 Reintegrate Merge

Use **Modify|Reintegrate Merge** to “reintegrate” changes from another URL to the selected file/directory.

Reintegrate merging is different from the “normal” merging: It carefully replicates only those changes unique to the source **Merge From** compared to the local working copy.

Example

You will typically use **Reintegrate Merge** after the work in a *feature branch* has been finished and the “feature” is to be reintegrated into the *trunk*. Here it’s important that all the previous merges from *trunk* to the *feature branch* are filtered out to avoid unnecessary merge conflicts and other problems. That is in short what reintegrate does. For a detailed explanation, refer to <http://svn.haxx.se/users/archive-2008-05/0808.shtml>.

For details regarding the **Advanced** options, refer to Section 3.7.1.

3.7.4 Apply Patch

Use **Modify|Apply Patch** to apply a *patch file* to your working copy. Currently supported patch file formats are *unidiff* patches. See Section 3.10.10 on how to create patches with SmartSVN.

For the **Select Patch File** dialog, select the patch file which you want to apply. Typically, patch files have *.patch* or *.diff* extensions. Based on the file paths contained in the patch file, SmartSVN will try to detect the correct base directory to which the patch should be applied. It will fail if it can’t find any files to patch in the working copy.

The resulting window is similar to the Merge Preview (see 7.9) window. Refer to that section for a description of the available commands. The **Files** area allows you to deselect certain files from the patch. You can apply the patch via **Patch|Apply Patch**.

Unpatchable files

In case the patch could not be applied to certain files, an **Unpatchable files** area will be displayed at the top of the window. The table contains the **Path** of the file and a description of the **Problem**. The tooltip text of the **Problem** column contains more details in case the expected and the actual lines did not match when trying to apply the patch to the file.

3.8 Properties

Both files and directories can have properties attached to them. There exists a set of predefined properties, which are used by SVN itself to manage the working copy. All other properties are “user-defined” properties. The following commands are related to properties and can be found in the **Properties** menu.

3.8.1 Edit Properties

Use **Properties|Edit Properties** to display and edit properties of the selected file/directory. For details refer to Section 7.2.

Note Internal SVN properties are displayed with grey font. It's not recommended to modify SVN properties directly through this dialog. It's better to use the other property-specific commands in the **Properties** menu.

You can **Add**, **Edit** and **Remove** individual properties. Use **Revert** on one or more properties to reset their **Current Value** to their **Base Value**.

3.8.2 Set or Delete Property

Use **Properties|Set or Delete property** to change a property for multiple files/directories at once.

Enter the name of the **Property**; the drop-down button offers the SVN internal properties for selection. To set a property value, either select **Set Value To** and enter the new property value, or, in case of boolean SVN-properties, select **Set boolean property**. To remove the property, select **Delete Property**.

For directories, choose to **Recurse into subdirectories** and optionally to **Include this directory**. Choose **Force** to skip a couple of checks which are performed for certain property (values).

Example

To get rid of all *explicit mergeinfo* from your project except from the project root, select `svn:mergeinfo` for **Property**, choose **Delete Property** and **Recurse into subdirectories** and deselect **Include this directory**.

3.8.3 MIME-Type

Use **Properties|MIME-Type** to change the *SVN MIME-type* of the selected files. The MIME-type can be either a default **Text**, a default **Binary** or a **Custom** type. In case of a **Custom** type, you have to specify the corresponding MIME-type here. E.g. “text/html”, “application/pdf” or “image/jpeg”.

MIME-types can't be arbitrary strings but must be *well-formed*. For instance, a MIME-type must contain a “/”. By default, SmartSVN checks whether MIME-types are well-formed. Use **Force** to disable this check.

The MIME-types are relevant for some SVN operations, for instance updating, where in the case of *text* types, the line endings etc. can be replaced. By default, when adding files (see Section 3.5.1), the coarse MIME-type (either *text* or *binary*) is automatically determined by SmartSVN. In general this detection is correct, but in certain cases you may want to explicitly change the MIME-type of the file with this command.

Within the project settings (see 6.3.3) you can define file name patterns which should always be treated as *binary*.

3.8.4 EOL-Style

Use **Properties|EOL-Style** to change the *EOL-Style* (line separator) of the selected files. The EOL-style is used when updating or checking out a text file and results in a corresponding conversion of its line endings:

- **Platform-dependent** converts to the platform's native line separators.
- **LF, CR, CR+LF** converts to the corresponding line separators, regardless of the current platform.
- **As is** performs no conversion.

In the project settings (see 6.3.3), the default EOL-style can be specified. This will be applied to every added file. By default, this will be **Platform-dependent**.

When changing the EOL-style of a file, SmartSVN checks whether the file has consistent line endings. If this is not the case, it will reject the EOL-style change (other behaviors can be configured in the project settings). To skip this check, use **Force**.

3.8.5 Keyword Substitution

Use **Properties|Keyword Substitution** to select the keywords for the selected files, which should be substituted (expanded) locally. Keyword substitution only works for text files.

For each keyword you have the option to **Set** or **Unset** it. Select **Don't change** to keep the current substitution for the keyword.

3.8.6 Executable-Property

Use **Properties|Executable-Property** to change the “Executable-Property” of the selected files. The “Executable-Property” is a versioned property, but is only used on Unix(-like) platforms, where it defines whether the “Executable Flag” should be set to a file or not.

Choose **Executable** if the “Executable-Property” should be assigned to the file or **Non-Executable** to remove the property from the selected files.

3.8.7 Externals

Use **Properties|Externals** to define or change externals. An *external* (officially also referred to as *externals definition*) is a mapping of a **Local Path** to an **URL** (and possibly a particular **Revision**) of a versioned resource.

In general, externals are specified by complete URLs, but there are also shorter representations which can be more flexible. The **URL** input field allows switching between the available representations for a given URL. For a detailed description of externals and valid URL formats, refer to <http://svnbook.red-bean.com/nightly/en/svn.advanced.externals.html>.

Example

To include the external `http://server/svn/foo` as directory `bar/bazz` at revision 4711 into your project, select directory `bar` and invoke **Properties|Externals**. Click **Add**, enter `bazz` into the **Local Path** input field, `http://server/svn/foo` into the **URL** input field, 4711 to the **Revision** input field and confirm by hitting **OK**: After committing your property change, an update on `bar` will create the subdirectory `bar/bazz` with the content from `http://server/svn/foo` at revision 4711.

Tip	It is safer to always set a Revision to externals. In this way you can always be sure about which version you are actually working with. When you decide to use a more recent revision of the external, you can evaluate it beforehand and, if you are satisfied, increase the Revision number of the external definition.
------------	--

Note	Externals may refer to directories as well as to files. In case of files, the referred URL must be part of the same repository to which its local parent directory belongs. (The local parent the directory is the directory to which the <code>svn:externals</code> property belongs.)
-------------	---

3.8.8 Ignore Patterns

Use **Properties|Ignore Patterns** to add, change or delete *local ignore patterns* for a directory. *Local ignore patterns* define file and directory patterns to be ignored within the directory.

Local ignore patterns are stored within the working copy (in the `svn:ignore` property of the directory) and will be committed. Therefore ignore patterns can only be applied to versioned directories.

By default, the **Patterns** are only set to the selected directory. You may also choose to set the patterns to all subdirectories by **Recurse into subdirectories**. In case of recursive ignore patterns, you may alternatively consider specifying *global ignore patterns* within the project settings (see 6.3.3).

To add an ignore pattern, you can also use the **Modify|Ignore** command.

3.8.9 Bugtraq-Properties

Use **Properties|Bugtraq-Properties** to configure the *Bugtraq-Properties* for the current working copy. Bugtraq-Properties are a technique for integrating Subversion with issue tracking systems.

A detailed specification for the *Bugtraq-Properties* can be found at: <http://tortoissvn.tigris.org/svn/tortoissvn/trunk/doc/issuetrackers.txt>, username is `guest` with empty password. SmartSVN implements this specification with the following mapping from UI elements to core `bugtraq:-properties` as shown in Table 3.1.

bugtraq-Property	UI Element
bugtraq:url	URL
bugtraq:warnifnoissue	Remind me to enter a Bug-ID
bugtraq:label	Message Label
bugtraq:message	Message Pattern
bugtraq:number	is true exactly if Bug-ID is set to Numeric
bugtraq:append	is true exactly if Append message to set to Top
bugtraq:logregex	For the version with one regular expression this corresponds to Bug-ID Expression . For the version with two regular expressions, Message-Part Expr. corresponds to the first line and Bug-ID expression corresponds to the second line.

Figure 3.1: Mapping from core bugtraq:properties to SmartSVN UI elements

Example

Assuming your commit messages looks like this: Ticket: 5 Some message or ticket #5: Some message and you want the 5 to be rendered as a link to your issue tracker. In this case, set **Bug-ID Expression** to [Tt]icket:? #?(\d+) and leave **Message-Part Expr.** empty.

If you want the whole Ticket #5 part to show up as a link, use the same **Bug-ID expression** and also set **Message-Part Expr.** to this value.

Example

Let's say your commit messages look like this: CF-11: Some message, or ET-12: Some message. Then, if you want the 11 and 12 to show up as links to your issue tracker, set **Bug-ID Expression** to \d+ and the **Message-Part Expr.** to (CF|ET)-(\d+).

If you want the whole CF-11 or ET-12 part to show up as a link, set **Bug-ID expression** to (CF-\d+|ET-\d+) and leave **Message-Part Expr.** empty.

3.8.10 Merge Info

Use **Properties|Merge Info** to change the svn:mergeinfo property for the selected files/directory.

Warning! The svn:mergeinfo is a core part of Subversion's *merge-tracking* mechanisms and is automatically managed by **Modify|Merge** and related commands. If you want to manually “block” certain revisions from being merged, you should use **Modify|Merge** with the **Record only** option set.

3.9 Tags and Branches

SmartSVN simplifies the handling of “Tags” and “Branches”. Both “Tags” and “Branches” are not native SVN concepts, but can easily be handled by the help of Copy To Repos-

itory (see 3.5.11) and Copy Within Repository (see 3.5.12). SmartSVN provides special support for managing tags and branches, which are based upon these copy commands.

Commands related to the management of tags and branches are available from the **Tag+Branch** menu. Various other commands support tags and branches alternatively for entering raw URLs.

3.9.1 Tag-Branch-Layout

The *Tag-Branch-Layout* defines the project's root URL (within the repository) and where the *trunk*, *tags* and *branches* of the project are stored. For various commands this will affect the presentation, and interaction, of the URLs. When invoking a tag/branch-aware command on a directory for which no layout can be found, SmartSVN will prompt you to configure a corresponding layout in the **Configure Tag-Branch-Layout** dialog.

A Tag-Branch-Layout is always linked with a corresponding **Project Root**. A **Project Root** is simply the URL of the top-most directory of a *project*. Any directory can be defined as a *project root* as the definition of what a *project* is, is completely up to you.

The first decision for a **Project Root** is whether to enable or disable Tag-Branch-Layouts for it. Many SVN projects are organized using tags and branches. In this case choose **Use the following layout** to configure the layout. If the corresponding project is not organized by tags and branches, choose **Do not work with tags and branches for this project root** to switch Tag-Branch-Layouts off.

Trunk specifies the root directory of the project's trunk. **Branches** and **Tags** specify the directory patterns of the branch and tag root directories, respectively. All paths are relative to the **Project Root**. Enter the values `trunk`, `branches/*` and `tags/*` here if you want to use the recommended SVN standard layout.

Example

The Subversion project itself is located at <http://svn.collab.net/repos/svn/>. Hence for the corresponding SmartSVN project, **Project Root** must be set to <http://svn.collab.net/repos/svn/>. Subversion's Trunk URL is <http://svn.collab.net/repos/svn/trunk>, i.e. `trunk` is the relative path and must be set for **Trunk**. Branches are located in <http://svn.collab.net/repos/svn/branches>, e.g. <http://svn.collab.net/repos/svn/branches/1.5.x> is the root of the "1.5.x" branch. I.e. **Branches** must be set to `branches/*`. This is similar for **Tags**.

It's also possible to use multiple branch or tag patterns. In this case, when entering, for example, a branch, you have to specify not only the branch name, but the relative path to the common root of all branches.

Example

A project may also contain *shelves* which can be interpreted as “personal branches”. For instance, the **Project Root** is located at `svn://server/svn/proj`. The “normal” branches are located in `svn://server/svn/proj/branches` and the shelves are located in `svn://server/svn/proj/shelves/[username]`, e.g. `svn://server/svn/proj/shelves/bob/my-shelve`. Hence, for **Branches** the following patterns should be used: `branches/*`, `shelves/*/*`.

Now, when creating a branch e.g. “b1” with **Tag+Branch|Add Branch**, you have to enter `branches/b1`, so SmartSVN knows that the branch should be created in the `branches` directory.

For example, when switching to Bob’s “my-shelve” with **Modify|Switch**, you have to enter `shelves/bob/my-shelve`, so SmartSVN knows that it should switch to a branch within the `shelves/bob` directory.

SmartSVN uses the proposed standard layout for new projects. If you want to configure another default layout, open one project which contains the desired layout, select **Tag+Branch|Configure Layout** and use **Make this configuration the default** here.

3.9.2 Add Tag

Use **Tag+Branch|Add Tag** to create a copy (“Tag”) of your local working-copy in the `tags` directory of your repository. **Name** will be the name of the tag and **Location** shows the corresponding location. You can create two kinds of tags:

- **Working Copy** tags are a snapshot of your current working copy. If the latter contains local changes, you will be asked whether these local changes should be included in the snapshot. The local changes will also reflect mixed local revisions and switched directories.
- **Repository Revision** tags are “server-side” tags which represent a snapshot of the repository at a given revision.

Tip **Repository Revision** tags can be useful if your working copy contains local changes but you don’t want them to be part of the tag. However, in this case you should make sure that your working copy actually corresponds to the revision which you plan to tag, i.e. you should do an update (see 3.4.1) to that revision beforehand and make sure that there are no switched directories.

By default, SmartSVN will **Abort** if the specified tag already exists. Select **Replace existing one** to create the tag anyway, replacing the existing tag.

Use **Externals Revisions** to specify how to handle externals revisions (see 3.8.7). For details refer to Section 3.5.11.

Note This command is similar to **Modify|Copy Local to Repository** (see Section 3.5.11), but is tailored to the special case of “Tagging”.

3.9.3 Tag Multiple Project Roots

Use **Tag+Branch|Tag Multiple Project Roots** on one or more project roots (working copy roots) to create a tag for all of these roots.

Enter the **Tag Name** and **Commit Message** which will be used for the creation of the tag. Select **Fix external revisions** to have all revisions of externals set to their current values, as present in the working copy.

This functionality is provided by the Tag Multiple Project Roots plugin (see 10.9).

3.9.4 Add Branch

Use **Tag+Branch|Add Branch** to create a copy (“Branch”) of your local working-copy in the `branches` directory of your repository. This command is similar to **Tag+Branch|Add Tag**, refer to Section 3.9.2 for details.

3.9.5 Tag Browser

Use **Tag+Branch|Tag Browser** to display all tags and branches of your project in a hierarchical structure. The hierarchy denotes which tags/branches have been derived (i.e. copied) from other branches.

Tags and **Branches** display the tags or branches location as specified with the **Configure Layout** (see 3.9.6) command. The subsequent table will contain tags and branches found herein. A tag or branch has a **Name**, a **Revision** at which it had been created and optionally a **Removed At** revision at which it had been removed.

The tag browser is built upon information from the Log Cache (see 5.3). With **Refresh** you can refresh the cache and rebuild the tag/branch-structure.

Tags/branches can be deleted by **Remove** which will remove the corresponding directory from the repository.

From the **Options** button you can select to show both **Branches and Tags**, **Branches only** or **Tags only**. **Recursive View** specifies whether the table shall also display tags/branches which have been *indirectly* derived from the currently selected branch in the tree. Select **Removed Tags and Branches** to also display tags/branches which have been deleted within the Repository. The corresponding items will contain a red minus within their icon to denote the deletion.

The **Branch** drop-down button allows sorting of the branches either **by Name** or **by Revision**.

Tip You can also invoke the Tag Browser from the tag or branch name input fields by clicking the ellipsis button to the right (...) or using `<Ctrl>+<Space>`-keystroke.

3.9.6 Configure Layout

Use **Tag+Branch|Configure Layout** to configure the *Tag-Branch-Layout* for the currently selected directory. This command is only available on the working copy root directory and externals roots (see 3.8.7). For details refer to Section 3.9.1.

Select **Make this configuration the default** to have this layout applied to all new projects.

3.10 Queries

SmartSVN offers following non-modifying commands – some of them work locally, others by querying the repository – from the **Query** menu.

3.10.1 Show Changes

Use **Query|Show Changes** to compare the selected files or directory against their pristine copies. **Show Changes** will correspondingly open one or more File Compare (see 7.1) frames or the Properties Compare (see 7.2) for a directory. For details, regarding the warning limit on the number of files to compare at once, refer to Section 8.7. No connection to the repository is required.

3.10.2 Compare with HEAD

Use **Query|Compare with HEAD** to compare a single, local file with the HEAD revision in the repository. If you want to compare against an arbitrary revision or some other repository file, use Compare with Revision (see 3.10.4).

3.10.3 Compare with Previous

Use **Query|Compare with Previous** to compare a single, local file with the last but one revision in the repository (i.e. the revision before HEAD). If you want to compare against the HEAD revision itself, use Compare with HEAD (see 3.10.2). If you want to compare against an arbitrary revision or some other repository file, use Compare with Revision (see 3.10.4).

3.10.4 Compare with Revision

Use **Query|Compare with Revision** to compare a single, local file with another revision of the same file or even another file. Select either to **Compare** the **Working copy** or the **Pristine copy**. Select to compare **With** the **Trunk** or a specific **Branch** or **Tag** or an arbitrary **Other Location**. Select whether to retrieve the repository file **At** the repository **HEAD** or at a another **Revision**. The result will be a File Compare (see 7.1) frame.

Tip	Use Compare with HEAD (see 3.10.2) if you want to quickly compare a file against the latest repository revision.
------------	--

3.10.5 Compare 2 Files

Use **Query|Compare 2 Files** to compare two local files with each other. No connection to the repository is required.

When having one or more *missing* files selected, their pristine copies will be used for the comparison instead.

3.10.6 Compare Repository Files or Directories

Use **Query|Compare Repository Files or Directories** to compare two different repository directories for changes (either added, removed or changed files and directories). This command gives you similar information like Create Patch between URLs (see 3.10.11), but in an easier to read form. The result will be a Compare Repository Files or Directories (see 7.3) frame.

The comparison is performed for one **Repository** between directories **From** and **To**.

Select **Recurse into subdirectories** to compare not only the directory and its immediate files, but also descend into subdirectories. Regarding **Ignore Ancestry**, refer to Section 3.10.11.

3.10.7 Log

Use **Query|Log** to display the Log (i.e. the commit history) of the selected file or directory. The Log command will open the Log window (see 7.6).

3.10.8 Revision Graph

Use **Query|Revision Graph** to open the Revision Graph for the selected file or directory. The Revision Graph essentially consists of a table of all revisions that make up the history of the selected file or directory, and a graph on the left side of that table which shows the parent-child relationships between the revisions. Included in the graph is the display of branching and merging points.

What happens when you open the Revision Graph depends on the option **Show dialog to allow root selection** in the Revision Graph preferences, see Section 8.9. If enabled, opening the Revision Graph will lead you to a dialog where you can select the root directory for which the Revision Graph should be produced. Additionally, the dialog allows you to specify whether you want to include modified files and directories beneath the root directory in the Revision Graph, via **Report children**.

If the root selection dialog option is disabled, the root directory for the Revision Graph is determined as follows:

- What is included in the creation of the Revision Graph depends on the selection before the Revision Graph was opened. To show a Revision Graph for the entire project, select the project root before opening the Revision Graph. To show a Revision Graph for a certain file or directory, only select that respective file or directory.

- When opening a Revision Graph on a directory, you will be asked whether to create the Revision Graph for the selected directory only, or also for all modified files and directories beneath the selected directory.

After invoking the Revision Graph command, the **Revision Graph** window of the selected file or directory will show up. For details, refer to Section 7.7.

3.10.9 Annotate

Use **Query|Annotate** to view the “history” of the selected file on a per-line basis.

On the **Configuration** tab, you can specify the date range over which to run the Annotate operation.

On the **Advanced** tab, select **Treat even binary revisions as text (“force”)** to continue the Annotate operation even when it encounters one or more binary revisions of the file. This option can be necessary if the MIME-Type (see 3.8.3) of a file was changed, e.g. from *binary* to *text* in some earlier revision, and the file had *text* content throughout. In case the file actually had binary content in some earlier revision, parts of the annotate output might contain junk.

After performing this command, the **Annotate** window for the selected file will show up. For details refer to Section 7.8.

3.10.10 Create Patch

Use **Query|Create Patch** to create a “Unidiff” patch for the selected files/directory. A patch shows the changes in your working copy on a per-line basis, which can for instance be sent to other developers. See Section 3.7.4 on how to apply patches with SmartSVN.

The patch will be written to the given local **Output File**. In case of creating a patch for a directory, you can select **Recurse into subdirectories** to create the patch recursively for all files within the selected directory.

Select **Ignore changes in EOL-Style** to omit output for line changes in which only the line ending differs. This can be useful if, after having the line endings of a local file changed temporarily, only “relevant” changes should be included in the patch.

With **For Whitespaces** you can configure to omit output for certain changes which are only affecting whitespaces. Use **No special handling** include any changes regarding whitespaces. Use **Ignore changes in the amount** to ignore lines for which only blocks with one or more whitespace characters have been replaced by blocks with one or more other whitespace characters. Use **Ignore them completely** to only output lines where anything except whitespaces has changed.

3.10.11 Create Patch between URLs

Use **Query|Create Patch between URLs** to create a “Unidiff” patch between two arbitrary URLs. See Section 3.10.10 for more details on patches. Compare Repository Files or Directories (see 3.10.6) is a version of this command that presents the results in a more human-friendly way.

The patch is generated from one **Repository** and contains the difference between **From** and **To**. The patch will be written to the local **Output File**.

By default, this command takes the ancestry into account, meaning it does not simply calculate (and print out) the difference between two files which have the same path, but also checks if both files are actually related. You can switch this behavior off by selecting **Ignore ancestry** on the **Advanced** page. For details regarding the other **Advanced** options, refer to Section [3.10.10](#).

3.10.12 Export Backup

Use **Query|Export Backup** to export a backup of the selected files/directory.

Export displays what will be exported. Depending on the selection of files/directory this will either be the number of files being exported or **All files and directories**. **Relative To** displays the common root of all files to be exported and the exported file's paths will be relative to this directory.

You can either export **Into zip-file** or **Into directory**. In both cases, specify the target *zip* file or directory, and optionally choose to **Wipe directory before copying**.

Select **Include ignored files** and/or **Include ignored directories** if you want to include the respective ignored items (and their contents) as well.

3.10.13 Conflict Solver

Use **Query|Conflict Solver** to start a *Three-Way-Merge*, which can be invoked on *conflicting* files (see Table [2.5](#)). For details, refer to Section [7.4](#).

When invoking this command on a binary file, it will open the Mark Resolved (see [3.5.14](#)) dialog.

3.11 Locks

Since Subversion 1.2, explicit file locking is supported. File locking is especially useful when working with binary files, for which merging is not possible.

For each file, its lock state is displayed in the file table column **Lock** and additionally the **Name** icon can contain corresponding overlay icons, as shown in Table [2.7](#). For a list of possible lock states, refer to Table [3.2](#).

The “Self” state can be filled in by SmartSVN when scanning the local working copy. Please note that this state can change when scanning the repository (see Section [3.11.1](#)), as the lock might actually be “Stolen” or “Broken”.

3.11.1 Refresh

With **Locks|Refresh** SmartSVN will scan the selected files, or all files within the selected repository directory, for locks. The result is displayed in the file table column **Lock**. This column is automatically made visible, if necessary.

Name	Meaning
(Empty)	The file is not locked.
Self	The file is locked for the local working copy.
Stolen	The file was locked for the local working copy but in the meantime it has been stolen by someone else in the repository.
Broken	The file was locked for the local working copy, but in the meantime it has been unlocked (by someone else) in the repository.
(Username)	The file is currently locked by the named user.

Figure 3.2: Lock States

You can combine scanning the repository for locks with refreshing the Remote State (see 3.12) in the Preferences (see 8.8). You can also schedule a repeated refresh of the repository lock information in the Project Settings (see 6.3.3).

3.11.2 Lock

With **Locks|Lock** you can lock the selected files in the repository. You can also enter a **Comment** to explain why you are locking these files.

The option **Steal locks if necessary** will lock the requested files regardless of their current lock state (in the repository). With this option it may happen that you “steal” the lock from another user, which can lead to confusion when the other user continues working on the locked file. Hence you should only use this option if necessary (e.g. if someone is on holiday and has forgotten to unlock important files).

Keep **Update to HEAD before** selected to perform an update to HEAD. Only the latest revision of a file can be locked.

3.11.3 Unlock

With **Locks|Unlock** you can unlock the selected files, or all files within the selected directory (recursively) in the repository.

The option **Break locks** will unlock the requested files even if they are not locked locally. With this option it may happen that you “break” the lock of another user, which can lead to confusion if that other user is still working on the locked file.

3.11.4 Show Info

Locks|Show Info shows information on the lock state (in repository) of the selected file.

State shows the current lock state (see Table 3.2). **Token ID** is the SVN Lock Token ID, which is normally not relevant for the user. **Owner** is the name of the user who currently owns the lock. **Created At** is the time when the lock has been set. **Expires At** is the time when the lock will expire. **Needs Lock** indicates whether this file needs locking, i.e. the “Needs Lock” property has been set (see Section 3.11.5). **Comment** is the lock comment, as entered by the user at the time of locking.

3.11.5 Change 'Needs Lock'

With **Locks|Change 'Needs Lock'** files can be marked/unmarked depending on whether they require locking. This is a useful indicator to users that they should lock the file before working with it. One aspect of this indication is that SmartSVN will set files which require locking (due to this property) to read-only when checking out or updating.

3.12 Remote State

The remote state shows the files' repositories states compared to the local working copy. It can also be interpreted as the action that would happen when updating the working copy to HEAD (see Section 3.4.1). The remote state of files is displayed in the file table column **Remote State**, the remote state of directories is displayed in the tooltip for a directory. See Table 3.3 for the list of possible remote states of files and directories.

Name	Meaning
Unchanged	The local entry is equal to the latest revision of this entry in the repository. An update on this entry will bring no changes.
Modified	For the local entry there exists a newer revision in the repository. An update will bring the corresponding changes for this entry.
Removed	The local entry has been removed in the repository. An update will remove the entry locally.
Added	This entry does not exist locally, currently in a versioned state. An update will add this entry.
Obstructed	For the local entry the latest repository revision contains another entry for being added. An update will fail here.

Figure 3.3: Remote State Types

To display the complete remote state information, especially the “Will be added” state, it may be necessary to add directories and files that do not exist locally to the directory tree or the file table. To such directories and files the special local state “Remote” is assigned, see Table 2.5 and Table 2.1.

3.12.1 Refresh Remote State

With **Query|Refresh Remote State** SmartSVN will query the repository and compare the latest repository revision with your local working copy. In this way, for each file and directory the corresponding remote state is assigned and displayed in the **Remote State** column; it will be made visible if necessary.

Refresh Remote State can be combined with the local Refresh and the scanning for locks (see 3.11.1) in the Preferences (see 8.8) to have the Remote State automatically refreshed.

If problems during the Remote State refresh are encountered, the status bar (see 2.1) will display an **Error** in the *Refresh* area. The tooltip for this area will show more details regarding the encountered problem.

3.12.2 Clear Remote State

Use **Query|Clear Remote State** to clear and hide the remote state. This will remove all directories and files which have the local state “Remote” (see Table 2.5 and Table 2.1) and hide the **Remote State** file table column.

3.13 Change Sets

A Change Set is a group of committable files and directories, with a message assigned. Subversion itself supports *Changelists* which currently can contain only files. SmartSVN automatically synchronizes the files of a Change Set with the corresponding SVN changelist. Change Sets are also known as “prepared commit” in other version control systems.

Change Sets are displayed in the Directory Tree (see 2.4) below the normal project directory structure. Table 3.4 shows the icons which are used for Change Set directories.




Icon	Description
	Change Set root node
	Change Set root node, which contains the modified project root directory
	A <i>virtual</i> Change Set directory, which does not represent an actual project directory, but is necessary to display child directories and files.
(various)	A Change Set directory which represents (or is equal to) the corresponding project directory, see Table 2.1.

Figure 3.4: Change Set icons

You can create a Change Set by selecting the files/directory to assign to the Change Set and invoking **Change Set|Move to Change Set** (Section 3.13.1). You can use the same menu item to add more committable files/directories to the Change Set, to move the selected files/directories to a different Change Set or to remove files/directories from a Change Set. When you are ready to commit, you can simply select the Change Set in the directory structure and invoke **Modify|Commit** (Section 3.6).

When the project directory structure is selected (as opposed to a Change Set), deactivating **View|Files Assigned to Change Set** (Section 2.4) will give a better overview of changed files not already assigned to a Change Set.

Note	A file/directory can only be assigned to one Change Set.
-------------	--

3.13.1 Move to Change Set

Use **Change Set|Move to Change Set** to change the assigned Change Set (see 3.13) of selected, committable files/directories.

To move the selected files/directory to a new Change Set, select **New Change Set** and enter the **Message** of the new Change Set. Select **Remove this Change Set once it gets empty** to automatically remove this Change Set once it gets empty. Select **Allow only committable entries** to automatically remove *unchanged* and other non-committable entries from Change Sets.

Example

When having **Remove this Change Set once it gets empty** and **Allow only committable entries** selected, the Change Set will be automatically removed after committing it because

- the committed files will turn their state into *unchanged* after the commit and hence will be removed from the Change Set and
- the Change Set will be empty and hence will be removed itself.

To move the selected files/directory to another, already existing Change Set, select **Existing Change Set** and choose the **Target Change Set**.

To remove the selected files/directory from their currently assigned Change Set, select **Remove from Change Set**.

Tip You can use Drag-and-Drop to move files to a Change Set.

3.13.2 Move Up

Use **Change Set|Move Up** to move the selected Change Set (see 3.13) one position up (when having multiple Change Sets).

3.13.3 Move Down

Use **Change Set|Move Down** to move the selected Change Set (see 3.13) one position down (when having multiple Change Sets).

3.13.4 Delete

Use **Change Set|Delete** to delete the selected Change Set (see 3.13). This will only affect the Change Set assignment, not the files nor their SVN state.

3.13.5 Edit Properties

Use **Change Set|Edit Properties** to change the **Message** and other properties of the selected Change Set (see 3.13). For details, refer to Section 3.13.1.

3.14 Common Features

SmartSVN includes a set of common features and UI elements that are shared by various commands.

3.14.1 Recursive/Depth options

In directory mode, most commands can work *recursively* and *non-recursively*. By default, SmartSVN offers a basic option **Recurse into subdirectories** (or a similar name) which let's you operate either only on the directory itself, or on the directory and all contained files and subdirectories, i.e. recursively.

Alternatively, you can switch to *advanced* recursion options in the Preferences (see 8.4). In this mode SmartSVN offers the Subversion *depth* levels:

- **Only this directory** only operates on the directory/file itself.
- **Only file children** operates on the directory and its directly contained files.
- **Immediate Children (files and directories)** operates on the directory, its directly contained files and subdirectories, but not on files or directories within these subdirectories.
- **Fully recursive** operates on the directory, contained files and subdirectories recursively.

Obviously, having **Recurse into subdirectories** selected is equivalent to depth **Fully recursive** while having **Recurse into subdirectories** deselected is equivalent to depth **Only this directory**.

3.14.2 Revision input fields

Most input fields for which you can enter a revision number, support a *browse* function, which can be accessed either by a **Select** button or by hitting `<Ctrl>+<Space>`-keystroke.

A dialog displaying all revisions for the selected file/directory will come up. It shows all revisions for which the directory has actually been affected and additionally all revisions which correspond to a specific tag, see Section 3.9 for further details. The **Revision** column shows the revision number or the corresponding tag. The other columns display the revision's **Time**, **Commit Message** and **Author**, respectively. The **Path** column shows the revisions's root location.

The displayed revisions are taken from the Log Cache (Section 5.3), so recent revisions might not be contained in the list. In this case you can use **Refresh** to update the Log Cache (and implicitly the displayed revisions) from the repository.

Browse Revisions at specifies the *peg* revision for the location to browse. In general **HEAD** should be sufficient for *alive* locations. Otherwise, you may select the corresponding **Peg Revision**.

Example

When merging (see 3.7.1) revisions from *replaced* (and hence *dead*) branches, it will be necessary to enter the correct **Peg Revision** to identify the branch.

3.14.3 Repository path input fields

Most input fields for which you can enter a repository path, support a *browse* function, which can be accessed by the **Browse** or by hitting `<Ctrl>+<Space>`-keystroke.

The Repository Browser (Section 4) will come up as a dialog. Depending on the command from which the browser has been invoked, you can either select a repository file and/or a repository directory.

For certain commands – where necessary – *peg*-revisions are supported. Peg-revisions specify the **URL** of a repository path. This can be helpful when working with paths which do not exist anymore in the repository. In SmartSVN, you can append a peg-revision to a path by prefixing it with a “@”.

Example

To specify a path “/project/path” at revision *91*, enter `/project/path@91`.

3.14.4 Tag input fields

Input fields, for which you can enter a tag, like when using Switch (Section 3.4.4), support a *browse* function, which can be accessed by the **Browse** button or by hitting `<Ctrl>+<Space>`-keystroke.

The Tag Browser (Section 3.9.5) will come up to let you select a branch or tag.

For certain commands – where necessary – *peg*-revisions are supported. For details refer to Section 3.14.3.

Example

To specify a tag “my-tag” at revision *91*, enter `my-tag@91`.

3.14.5 File/directory input fields

Input fields, for which you can enter a path to a file or directory, support a *browse* function, which can be accessed by selecting the **Choose** button or by hitting `<Ctrl>+<Space>`-keystroke.

Chapter 4

Repository Browser

The Repository Browser offers a direct view into the repository and basic commands to manipulate repository contents directly. The Repository Browser comes as a stand-alone frame. It can be invoked from within the Project Window (see 2) by **Query|Open in Repository Browser** or by **Window|New Repository Browser**. If a tray icon (see 9.8) is present the Repository Browser frame can be invoked by **New Repository Browser**. The Repository Browser can also be invoked from Project Window (see 2) commands via Repository path input fields (see 3.14.3) and commands like Check Out (see 3.1) or Import into Repository (see 3.2).

The Repository Browser displays the repository content with a **Directory** tree and a **File** table, similar to the Project Window (see 2). For details on the **File Filter**, refer to Section 2.4.2.

The repository file system is only scanned on demand. This happens when currently unscanned directories are expanded. The Tag-Branch-Layouts (see 3.9.1) will be used to display directory icons. Table 4.1 shows the possible directory states.

4.1 Repository menu

- Use **Open** to change the currently browsed repository. This command opens a dialog where you can enter a **Repository URL** to browse. It's recommended (though not necessary) to enter repository root URLs.
- Use **Show Revision** to change the currently displayed revision.
- Use **Check Out** to check out the currently selected directory. This will bring up a simplified **Check Out** wizard, for details refer to Section 3.1.
- **Manage Log Caches**, see Section 5.3.1.
- Use **Close** to close the frame.
- Use **Exit** to exit SmartSVN.








Icon	State	Details
	Default	An already scanned repository directory without special meaning.
	Unscanned	A not yet scanned repository directory.
	Root	A project root, according to some Tag-Branch-Layout (see 3.9.1).
	Trunk/Branch	A <i>trunk</i> or <i>branch</i> , according to some Tag-Branch-Layout (see 3.9.1).
	Tag	A <i>tag</i> according to some Tag-Branch-Layout (see 3.9.1).
	Intermediate	An intermediate directory according to some Tag-Branch-Layout (see 3.9.1). For instance the parent directory (container) of the <i>branches</i> .
	Error	An error has occurred while scanning the repository, only displayed for the root directory.

Figure 4.1: Directory States

4.2 Edit menu

- Use **Stop** to cancel the currently processing operation. This action might not be applicable for certain operations.
- Use **File Filter** to put the focus into the **File Filter** field.
- Use **Configure Layout** to configure the Tag-Branch-Layout (see [3.9.1](#)) for the currently selected directory.
- Use **Dismiss Layout** to dismiss the Tag-Branch-Layout for the currently selected directory. This can be useful to get rid of a “deeper” layout in favor of its parent layout.
- Use **Copy Name** to copy the name of the selected file/directory to the system clipboard. If multiple files are selected, all names will be copied, each on a new line.
- Use **Copy URL** to copy the URLs of the selected file/directory to the system clipboard. If multiple files are selected, all URLs will be copied, each on a new line.
- Use **Set Encoding** to configure the encoding which will be used when displaying file contents for the various **Query**-commands. Refer to Section [6.3.1](#) for details on when encodings will be applied. The encoding will be stored in the corresponding Repository Profile (see [8.3](#)).
- Use **Customize** to customize accelerators (see Section [8.18](#)).
- Use **Preferences** to show the application preferences (see Section [8](#)).

4.3 View menu

- Use **Refresh** to explicitly refresh the contents of the **Directory** tree and the **File** table from the repository.
- Select **Files from Subdirectories** to also view files from within subdirectories of the currently selected directory.

4.4 Modify menu

- **Create Directory**, see Section 4.4.1.
- **Remove**, see Section 4.4.2.
- **Copy**, see Section 4.4.4.
- **Move**, see Section 4.4.4.

4.4.1 Create Directory

Use **Modify|Create Directory** to create a new directory in the currently selected directory. Enter the **Directory Name** which may contain slashes (“/”) to create multiple directories at once.

Select **Create default project structure for trunk, branches and tags** to also create sub-directories **trunk**, **branches** and **tags** in the created directory.

Enter the corresponding **Commit Message** which is automatically suggested, as long as you haven’t manually modified it.

4.4.2 Remove

Use **Modify|Remove** to remove the currently selected directory(ies) or files from the repository. Enter a corresponding **Commit Message**, which is automatically suggested based on the selected directory/files.

4.4.3 Rename

Use **Modify|Rename** to rename the selected file or directory. This will open a dialog where you can enter the new name of the file or directory.

4.4.4 Copy/Move

Use **Modify|Copy** or **Modify|Move** to copy or move the selected files/directory to another location. Select **Copy** to only copy the files/directory or **Move** to additionally remove the copy sources afterwards.

Use **To** to copy the copy sources itself to the selected location. When having selected one file/directory the entered destination location must not yet exist. The last part of the

destination path will be the new name of the copied file/directory. When having multiple files selected, the files will be copied into the destination path.

Use **Contents Into** to copy the contents of the copy source into the selected location. This option is only available for copying directories. In either case, necessary parent directories will be created automatically.

Enter the corresponding **Commit Message** which is automatically suggested, as long as you haven't manually modified it. Select **After command execution show repository at HEAD revision** to reset the Repository Browser's revision to HEAD after having performed the copy or move command. This option is only available if the current revision is not set to HEAD and it is convenient to immediately see the copy results (in HEAD).

Tip	You can also use Drag-And-Drop to copy or move files and directories. This will open the same dialog with the corresponding paths pre-filled.
------------	---

4.4.5 Edit Properties

Use **Modify|Edit Properties** to open a dialog that allows you to edit the SVN properties of the selected file or directory. The dialog is described in more detail in Section 3.8.1. Additionally, there is a text field at the bottom where you can enter a commit message for the property changes to be committed.

4.5 Query menu

- Use **Open** to open the currently selected file. SmartSVN will check out the file to a temporary location and open it in the specified editor. For details refer to the corresponding Open (see 2.5.7) command in the Project Window (see 2).
- Use **Compare** on a selection of two files or two directories to compare their contents. For details refer to Section 3.10.1 and Section 3.10.6, respectively.
- Use **Log** to display the log for the currently selected directory or file. For details refer to Section 3.10.7.
- Use **Revision Graph** to display the revision graph for the currently selected directory or file. For details refer to Section 3.10.8.
- Use **Annotate** to display an annotated view of the currently selected file's content. For details refer to Section 3.10.9.
- Use **Save As** to save the contents of the selected file to a local file. Enter the **Target Path** and select whether to **Expands keywords** or leave them unexpanded (as they are in the repository).

4.6 Window menu

Refer to Section 2.5.12 for more details.

4.7 Help menu

Refer to Section [2.5.13](#) for more details.

Chapter 5

Transactions

The *Transactions* are a direct view into a repositories' *Log* which is continuously updated. The Transactions are primarily designed to keep you up-to-date on what has happened within repositories you are interested in, but also to allow similar powerful queries as the Log command (see 3.10.7) itself. The Transactions are integrated into the Project Window (see 5.2) and they come as a stand-alone Transactions frame (see 5.1).

5.1 Transactions frame

The Transactions frame can be invoked from within the Project Window (see 2) or from within the Repository Browser (see 4) by **Window|Show Transactions**. If a tray icon (see 9.8) is present the Transactions frame can be invoked by **Show Transactions**.

The Transactions frame can be used to observe multiple repositories at the same time. Every revision of every repository is represented by one line in the transactions tree, which can be expanded to see which files/directories have been affected by the corresponding revision.

Note	For repositories in an older format than Subversion 1.6, the received log data does not contain information on whether a changed entry is of file or directory type. Hence, all entries modified in a revision will be displayed using file icons (even if there are directories).
-------------	--

A revision line primarily shows the commit message of the corresponding revision and has a prefix which shows various properties of that revision:

- **Root:** displays to which repository the revision belongs. This column is only present if multiple repositories are observed, refer to Section 5.1.2 for details. The column may also contain the “project name”, appended after a colon (“:”). The “project name” is the last path component of the project root of the corresponding Tag-Branch-Layout (see 3.9.1).
- **Revision Number:** Displays the revision number.
- **Time:** Displays date and time of the revision. The used format can be changed in the Preferences (see 8.4).

- **Trunk/Branch/Tag:** displays the corresponding *trunk*, *branch* or *tag* to which the revision belongs, refer to Section 3.9.1 for details. This column is only present if at least one of the displayed revisions actually belongs to a *trunk*, *branch* or *tag*.
- **Author:** Displays the revision’s author.
- **File count:** Displays the number of modified files/directories the revision contains.

The changed files/directories for a revision are displayed relative to the corresponding **Trunk/Branch/Tag** of the revision, or relative to the **Root**’s URL in case no Tag-Branch-Layout is used. If a Tag-Branch-Layout is used, but a file path does not fit into the Tag-Branch-Layout, it will be prefixed by a “/” to denote that it is given relative to the **Root**.

5.1.1 Grouping of revisions

Use the **View** to group the revisions by different categories:

- Ungrouped
- Days
- Weeks
- Date
- Authors
- Location (repository)

5.1.2 Watched URLs

Use **Edit|Configure Watched URLs** to configure the observed URLs (i.e. repositories). Every entry must have a **Name** which will be displayed in the “Root” column of the revision line prefix to distinguish revisions from different repositories. All revisions below the **Root URL** will be observed.

With the **Display revisions for the last** and **But at most** options you can put limits on how far into the past the Transactions view will display revisions.

Note For large and/or highly active projects, using a large value for **Display revisions for the last** without a reasonable **But at most** restriction can require significant memory usage and computational resources.

The watched URLs can be refreshed manually with **Transaction|Refresh** and they will be refreshed recurrently for the interval specified in the Preferences (see 8.13).




Icon	State	Details
	Default (read)	A (<i>read</i>) revision.
	Unread	An <i>unread</i> revision.
	Remote	A working copy revision which contains at least one file which will be updated when updating to <i>HEAD</i> .

Figure 5.1: Revision states

5.1.3 Read/Unread revisions

SmartSVN internally manages for every repository a list of which revisions are *Unread* and which revisions have already been *Read*. This mechanism is similar to how email clients work: Newly fetched revisions are considered as *Unread* and hence are displayed with a blue color. In addition to that, they will have a different icon. For details refer to Table 5.1. Use **View|Mark as Unread** or **View|Mark All as Read** to mark revisions as *unread* or *read*.

The read/unread state of revisions is not related to a single Transactions view, but shared by all views. For instance, multiple Project Window transactions (see 5.2) and the Transactions frame itself may show the same repositories. Marking a revision as read/unread will change their state in all of these views.

5.1.4 Display Settings

The layout of the revision line prefix can be configured in the display settings, via **View|Settings**. Choose whether to show **Time**, **Author**, **File/directory count** and/or **Trunk/Branch/Tag**.

5.1.5 Transaction menu

- Use **Refresh** to refresh the log information for the Watched URLs (see 5.1.2).
- **Manage Log Caches**, see Section 5.3.1.
- Use **Close** to close the frame.
- Use **Exit** to exit SmartSVN.

5.1.6 Edit menu

- Use **Stop** to cancel the currently processing operation. This action might not be applicable for certain operations.
- Use **Open** to open the currently selected file. SmartSVN will check out the file to a temporary location and will open it in the specified editor. For details refer to the corresponding Open (see 2.5.2) command in the Project Window (see 2).

- Use **Copy Message** to copy the commit message of the currently selected revision. If multiple revisions are selected, all messages will be copied, each on a new line.
- Use **Copy Name** to copy the names of the files modified in the currently selected revisions. In case of multiple files, each filename will occupy a separate line in the copied text.
- Use **Copy Relative Path** to copy the relative paths of the currently selected files. If multiple files are selected, all files will be copied, each on a new line.
- Use **Copy URL** to copy the URLs of the files modified in the currently selected revisions. In case of multiple files, each URL will occupy a separate line in the copied text.
- Use **Copy Revision Number** to copy the revision numbers of the files modified in the currently selected revisions. In case of multiple files, each revision number will occupy a separate line in the copied text.
- Use **Customize** to customize accelerators (see Section 8.18).
- **Configure Watched URLs**, see Section 5.1.2.
- Use **Preferences** to show the application preferences (see Section 8).

5.1.7 View menu

- **Mark as Unread**, see Section 5.1.3.
- **Mark All as Read**, see Section 5.1.3.
- **Ungrouped Revisions**, see Section 5.1.1.
- **Grouped by Days**, see Section 5.1.1.
- **Grouped by Weeks**, see Section 5.1.1.
- **Grouped by Date**, see Section 5.1.1.
- **Grouped by Authors**, see Section 5.1.1.
- **Grouped by Location**, see Section 5.1.1.
- **Settings**, see Section 5.1.4.

5.1.8 Modify menu

- **Change Commit Message**, see Section 7.6.4.

5.1.9 Query menu

- Use **Show Changes** to display the changes for the selected file or revision. For details refer to Section [3.10.1](#).
- Use **Log** to display the log for the currently selected revision or file. For details refer to Section [3.10.7](#).
- Use **Revision Graph** to display the revision graph for the currently selected revision or file. For details refer to Section [3.10.8](#).
- Use **Annotate** to display an annotated view of the currently selected file's content. For details refer to Section [3.10.9](#).
- Use **Save As** to save the contents of the selected revision/file to a local file, for details refer to (see [4.5](#)).

5.1.10 Window menu

Refer to Section [2.5.12](#) for more details.

5.1.11 Help menu

Refer to Section [2.5.13](#) for more details.

5.2 Project Transactions

The *Project Transactions* are displayed in the **Transaction** view which is by default located in the lower right area of the Project Window (see [2](#)). The Project Transactions view provides virtually all features of the stand-alone Transactions Frame (see [5.1](#)) and extends some of them.

Many commands available in the Project Transactions view are integrated into the various Project Window commands (see [2](#)), for instance Log (see [3.10.7](#)) transparently works on the the project files and directories as well as on Transaction revisions or files. The Transactions-specific commands can be found in the **Transactions** menu, see Section [2.5.11](#).

The main difference compared to the Transactions frame is that those revisions which are related to the current working copy (called *working copy revisions*) are implicitly displayed; similar to the Transactions frame further “watched URLs” can be configured by **Transactions|Configure Watched URLs**.

For working copy revisions, their read/unread (see [5.1.3](#)) state is tracked but not displayed in the Project Transactions. Instead, based on the local working copy state, the “remote state” for every revision is evaluated and displayed accordingly: If a revision has already been updated, it's simply displayed as *read*. If there is at least one file part of the revision which will be updated when updating (see [3.4.1](#)) to *HEAD* the revision is displayed as *read*, containing a *green arrow*, see Table [5.1](#).

5.2.1 Settings

Select **Transactions|Settings** to configure the Project Transactions.

Select **Repeatedly refresh transactions** to refresh the working copy transactions recurrently, with the same interval as for the Transactions frame. Select **Refresh after loading project** to automatically refresh the working copy transactions after a project has been loaded. Select **Refresh after a command changed the working copy** to automatically refresh after Updates, Commits, etc.

Regarding the basic **Display** options, refer to Section 5.1.4. **Display revisions for the last** and **But at most** refer to the working copy revisions; the meaning of these options is identical to the additionally watched URLs, for details refer to Section 5.1.2.

5.3 Log Cache

The Log Cache is the local data storage for the *Transactions*. It is also used by other SmartSVN commands, for instance the Log command (Section 3.10.7). It stores and supplies the raw log information as received from the server and can supply them to various commands later on. This can increase log performance significantly and also leads to reduced network traffic.

When *Log* information is requested for the first time for a certain repository, you can choose which parts of the repository should be indexed by the Log Cache. In general it is recommended to select **Create cache for whole repository at** to let SmartSVN index the whole repository. The reason for this is that logs of a certain “module” can have links to other modules, due to the way Subversion’s *Copy* mechanism works. Sometimes repositories can be very large and you may be interested only in a few modules of the whole repository. In this case it may be more efficient to select **Create cache only for module at** and select the corresponding module. However, this can lead to incomplete logs due to the reasons stated above. For some repositories you might want to use create no Log Cache at all. In this case choose **Skip cache and perform logs directly**.

SmartSVN automatically keeps the Log Cache(s) up-to-date. All log-related commands always query the repository for the latest logs, before querying the Log Cache. In the same way, every manually or automatically triggered refresh of the Transactions will update the corresponding caches.

Log results (for instance used by the Log command) from the Log Cache are in general identical to results obtained when querying the server directly. However there can be differences for following situations:

- Server-side access restrictions on already cached revisions are changed afterwards. This happens for instance, when using and modifying *AuthzSVNAccessFile* for *HTTP* repositories.
- Log information for already cached revisions are changed on the server afterwards. This happens for instance when changing the repository’s database directly or by changing *revision properties*, e.g. when another user has performed Change Commit Message (see 7.6.4).

In such cases, you should rebuild the Log Cache as described in Section 5.3.1.

5.3.1 Manage Log Caches

In the Project Window (see 2) use **Project|Manage Log Caches** to manage the local Log Caches.

The list shows all known **Root URLs** and the corresponding **Log Type**. For **Log Type** set to **Local Log Cache** there exists a local Log Cache for the **Root URL** against which logs will be performed. Otherwise, for **Direct Logs onto Repository**, the logs will be performed directly against the repository.

Log Caches are created on demand for a new **Root URL** and the choice whether to use a **Local Log Cache** or **Direct Logs onto Repository** has to be done when a log is first requested for that URL. This choice will be remembered and typically doesn't need to be changed afterwards. If necessary, you can use **Delete** for the corresponding **Root URL**. This will discard the **Log Type** choice and get rid of the Log Cache in case of **Local Log Cache** choice. Hence, subsequent log requests for this URL (or child URLs) will bring up the Log Cache initialization dialog again.

Select **Rebuild** for a **Local Log Cache** to rebuild it from repository log information. In general it's recommended to rebuild caches completely by selecting **All** unless you know that only log information **Starting with** a certain revision had been changed.

5.3.2 Storage

The Log Cache information is stored in the subdirectory **log-cache** in SmartSVN's settings directory. For every Log Cache, there is a separate subdirectory containing the server name and repository path. This is typically sufficient to quickly locate the cache for a specific repository. In case there are multiple subdirectories with the same name, only differing in the trailing number, you can have a look at the contained **urls** files. They show the exact location for which the Log Cache has been built.

If you encounter problems when rebuilding the cache, or if you need to get rid of the cached information for a certain repository, you can remove the corresponding subdirectory. Alternatively, you can remove the whole **log-cache** in order to get rid of all cached log information. You should never change these files while SmartSVN is running, otherwise the results will be unpredictable.

Chapter 6

Projects

SmartSVN internally manages your SVN working copies in “SmartSVN projects”. A SmartSVN project points to one or more SVN working copies (local SVN-controlled directories) and has a name and settings (Section 6.3) attached to it. When working with SmartSVN, you are always working with a project.

Projects can be created in different ways from the **Project** menu. To create a completely new project from a not-yet-version-controlled local directory, use **Import Into Repository** (see Section 3.2). This will also create the corresponding directory (module) in the repository. If you want to create a local working copy from a project which already exists in a repository, use **Check Out** (see Section 3.1).

6.1 Managing working copies

To create a project from an already versioned local directory, use **Project|Open Working Copy** and specify the local **Versioned Directory**. On the **Project** page, you may select to **Open in new project** for this working copy, specify the project’s name and specify an optional group (see Project Manager (see 6.2)) to which the project will be added. You may select **Add to current project** to add the working copy to the currently open project (if present). If there is already a project which contains this working copy, you may select **Open existing project** to open this project. Or you may select **Don’t manage as project** to just create a temporary project for this working copy.

If the location of a working copy has changed, you may use **Edit Working Copy** to point to the new location. To remove a working copy from the project, use **Remove Working Copy**.

Note	For an advanced configuration of the working copy roots use the project settings (see 6.3).
-------------	---

One Project Window shows one project at a time. To work with multiple projects at the same time, you can open multiple Project Windows by clicking **Window|New Project Window**. Already existing projects can be opened in a Project Window with **Open or Manage Projects** and closed with **Close**.

6.2 Project Manager

With the Project Manager (**Project|Open or Manage Projects**) you can manage your existing SmartSVN projects. The set of managed projects is arranged in a tree-structure. This allows you to group related projects under a common group name, etc.

Tip There is one special group **Sorted project area** which receives all new projects. This group is sorted and hence works like a sorted project list. If you don't need to group projects, simply let this group maintain the project list for you.

With **Rename** you can change the **Name** of an already managed project or a group. Use **Delete** to remove projects from the project tree; neither the local directory itself nor any other filesystem content will be affected by this operation.

You can rearrange the entries in the project tree with Drag-and-Drop. If a group is expanded, you can move the currently selected item into this group, otherwise it will be moved across.

Use **Create Group** to wrap the currently selected project in a group. Thereafter you can move other projects into this group. If you **Delete** a group, only this group will be deleted. Projects or groups contained within the group will not be deleted.

6.3 Project Settings

The project settings affect the behavior of various SVN commands. Contrary to the global preferences (see Section 8), the project settings only apply to an individual project. You can edit the settings of the currently opened project via **Project|Settings**.

The top of the dialog shows the **Root Paths** for the current project. Use **Change** to modify these paths to, for example, add other root directories, or to change a root directory after the corresponding SVN working copy has been moved on your local disk.

6.3.1 Text File Encoding

The text file encoding affects only the presentation of file contents, for instance when comparing a file (see Section 3.10.1) and it will only be used if for the file itself no *charset* has been specified by its MIME-Type property (see 3.8.3). The text file encoding settings are not relevant for SVN operations itself, which generally work only on the *byte*-level.

With **Use system's default encoding**, SmartSVN will automatically use the system's default encoding when displaying files. When changing the system encoding later, the project settings are automatically updated.

Alternatively you can choose a fixed encoding with **Use following encoding**.

6.3.2 Scan

The **Scan** settings specifies whether SmartSVN scans the **Whole project** or **Only root directory** when opening a project.

General we recommend using the **Whole project** option, because features like searching files in the table, etc. rely on having the whole project structure in memory. Nevertheless, when you are working with *large* projects, it can be necessary to scan the file structure on demand to avoid high memory consumption.

6.3.3 Working Copy

The option **(Re)set to Commit-Times after manipulating local files** tells SmartSVN to always set a local file's time to its internal SVN property `commit-time`. Especially in case of an updating command (see Section 3.4), this option is convenient to get the actual change time of a file and not the local update time.

Apply auto-props from SVN 'config' file to added files tells SmartSVN to use the *auto-props* from the SVN 'config' file, which is located in the `Subversion` directory below your home directory. These auto-props will also override other project defaults, like **Default EOL Style**, explained below.

Choose **Keep input files after merging** to always keep the auxiliary files (*left*, *right* and *base*) after a file has been merged by the Merge (see 3.7.1) or by the Merge from 2 sources (see 3.7.2) command. These files will be put into *merged* state (see Table 2.5) which is similar to the *conflict* state however without having actual conflicts. For *merged* files you can use the Conflict Solver (see 7.4) to review merge changes in detail and you can finally use Mark Resolved (see 3.5.14) to mark the file as resolved and to get rid of the auxiliary files.

Global Ignores

The Global Ignores define *global ignore patterns* for files/directories which should in general be ignored within the current project. This is contrary to local ignores (see Section 3.8.8), which are only related to a specific directory. You can completely deactivate Global Ignores by **Deactivated**. With **Use from SVN 'config' file**, the same ignore patterns will be used as by the command line client. Independently of the command line client, you can enter your own patterns via **Use following patterns**. The **Patterns** are file name patterns, where "*" and "?" are wildcard symbols, interpreted in the usual way.

Binary Files

Choose **Use MIME-type registry from SVN 'config' file** to use the corresponding file which is also used by the command line client. Choose **Use following patterns** to specify custom **Patterns**, for which matching files will always be added (see 3.5.1) with *binary* MIME-type (see 3.8.3). The wildcard symbols "*" and "?" can be used in the usual way.

EOL Style

This option specifies the **Default EOL-Style**, which is used when adding a file (Section 3.5.1). For more details refer to Section 3.8.4.

Use **In case of inconsistent EOLs** to configure the behavior when adding a file with inconsistent EOLs (line endings). **Add 'As Is'** will automatically add the file with EOL

style “As Is”. **Add as Binary** will automatically set the file’s type to “Binary”, see also Section 3.8.3. **Report Error** will report an error.

EOL Style – Native

Usually text files are stored with ‘native’ EOL-Style (see 3.8.4) in the SVN repository. As a result, after performing SVN operations on these files your platform’s native line separator will be used (‘Platform-dependent’). Under certain circumstances it can be convenient to redefine what ‘native’ means, e.g. when a project is operated on Windows OS, but frequently uploaded to a Unix server. Re-defining can be done here by choosing the desired **Native EOL-Style**.

Locks

Use **When adding files, set ‘Needs Lock’ for** to specify for which files the Needs Lock (see 3.11.5) flag should be set when they are added. With **No file**, the ‘Needs Lock’ property will be set on no file. With **Binary files** the property will only be set on files which have been detected to have binary content. With **Every file** the property will be set on every file.

When committing (see 3.6) files or directories, SmartSVN will scan for locked files. Choose here whether to suggest to **Release Locks** or to **Keep Locks** for those files on the “Locks” page of the commit wizard.

Enable **Automatically scan for locks** and enter the corresponding interval in **minutes** to repeatedly refresh the files’ lock states. Refer to Section 3.11.1 for details.

Keyword Subst.

This option specifies the Keyword Substitution default, which is used when adding a file (Section 3.5.1). For more details refer to Section 3.8.5.

Conflicts

By default, conflicting files will receive new extensions like “mine” or “.r4711”. Here you can specify extensions which should be preserved in case of conflicts. Choose either **Use from SVN ‘config’ file** or **Use following extensions** and enter the file name **Extensions** which should be preserved.

6.3.4 Default Settings

Projects are created by various commands. For reasons of simplicity, in most of these cases, there is no configuration possibility for the corresponding project settings. Therefore you can specify default project settings (template settings), which will be applied to every newly created project. With **Project|Default Settings** you can configure the same properties as for a concrete project.

Chapter 7

Subwindows

Many commands are resulting in stand-alone sub-windows with their own functionality and purpose.

7.1 File Compare

The *File Compare* window shows the contents of two files, one in the left and one in the right area of the window. A File Compare is typically invoked via **Query|Show Changes** from the Project Window (see 2) but there are other ways to invoke a File Compare in SmartSVN. Together with a File Compare, a Properties Compare (see 7.2) can be invoked, if properties of the files to compare are different as well.

Note	Depending on your File Compare settings (see 8.11), performing a file comparison can also invoke an external file compare tool. This section refers only to the built-in <i>File Compare</i> of SmartSVN.
-------------	---

Depending on the source of the compared files (local working copy, repository), neither, only the right, or both editors in the File Compare window may be editable. Depending on the invoking command, when a *copied* file is compared and the copy source file is *removed*, the pristine copy of the source file will be used for the comparison, provided that its contents are available.

Tip	If the file compare refuses to compare a file because it's <i>binary</i> , check the corresponding MIME-Type (see 3.8.3) property. Regarding the used encoding, refer to Section 6.3.1. You can also configure to the set MIME-Type and auto-detect the type in the File Compare settings (see 8.11).
------------	---

7.1.1 Comparison

The file contents are compared line-by-line, where the underlying algorithm finds the minimum number of changed lines between the left and the right content. The differences between left and right content is highlighted by colored regions within the text views,

which are linked together in the center *Link Component*. The Link Component allows to take over changes from one side to the other, depending on which side is editable.

Tip When the mouse is positioned over the Link Component, you can use the mouse wheel to jump from difference to difference.

Depending on the Preferences (see 8.10), not only complete lines, but also the content within lines is compared if they are not too different. These comparison results are *inner-line* changes. You can take over such changes from left to right or vice versa with **Apply Left** and **Apply Right**, respectively, from the popup menu (invoked on the change itself).

The following sections describe the various entries in the menu of the **File Compare** window. Some of the available operations work on the *active* view, i.e. the view which has the focus and/or displays the cursor.

7.1.2 File menu

- Use **Save** to save changes to (both) file(s).
- Use **Export as HTML-File** to export the comparison to an HTML-file.
- Use **Close** to close the frame.

7.1.3 Edit menu

Contains common functions to alter the file contents and to find a certain text within the contents. Additionally:

- Use **Take Left Block** to take over the complete block below the cursor position from left to right. This may also remove or insert blocks in the right view.
- Use **Take Right Block** to take over the complete block below the cursor position from right to left. This may also remove or insert blocks in the left view.
- Use **Customize** to customize accelerators (see Section 8.18).

7.1.4 View menu

- Use **Refresh** to refresh the contents of the files from disk. This command is not applicable if both file contents are read-only.
- Use **Left Beside Right** to display left and right files side-by-side, which is the default.
- Use **Left Above Right** to display the left file above the right file. This can be convenient when having files with long lines.
- **Ignore Whitespace for Line Comparison**, refer to Section 7.1.8.

- **Ignore Case Change for Line Comparison**, refer to Section [7.1.8](#).
- Select **Synchronize Scrolling** to enable the synchronization of the scrolling states of the two editors.
- **Settings**, refer to Section [7.1.8](#).

7.1.5 Go To menu

- Use **Previous Change** to scroll to the previous difference within the active view, relative to the current cursor location.
- Use **Next Change** to scroll to the next difference within the active view, relative to the current cursor location.
- Use **Go to Line** to go to the specified **Line Number** within the active view.

7.1.6 Window menu

Refer to Section [2.5.12](#) for more details.

In the following, the File Compare settings are described, which can be accessed via **View|Settings**.

7.1.7 General Settings

The **Tab Size** specifies the width (number of characters) which is used to display a TAB character. With **Show whitespaces** whitespace characters will be displayed. With **Show line numbers** a line number gutter will be prepended.

Select **Remember as default** to have the selected options apply to all File Compare frames.

For basic settings regarding text components, refer to Section [8.10](#).

7.1.8 Compare Settings

If **Ignore whitespace for line comparison** is selected, two lines are treated as equal, if they only differ in the number, but not in the position of whitespaces.

If **Ignore case change for line comparison** is selected, uppercase and lowercase characters are treated as equal.

The **Inner-Line Comparison** specifies the “tokenizing” algorithm of the lines, for which the individual tokens within two lines will be compared against each other. **Alphanumeric words** results in tokens, which form alphanumeric words, i.e. words, which are consisting only of letters and digits and which are starting with a letter. All other characters are considered as tokens on their own. **Character-based** treats every character as a single token. This is the most fine-grained comparison option. **C identifiers** and **Java identifiers** are similar to **Alphanumeric words**, but in addition to letters and digits certain other characters are allowed to be part of a single token. **Off** completely disables the inner line comparison, i.e. every line is considered as single token.

With **Trim equal start/end of Inner-Line changes** selected and two tokens being different, the equal starting and ending characters within both tokens won't be displayed. For instance, for the tokens `foobar` and `foupar` the difference will only display as `up`.

Select **Remember as default** to have the selected options apply to all File Compare frames.

7.2 Properties Compare

The *Properties Compare* window shows the properties of two files or directories, one in the left and one in the right area of the window. A Properties Compare is typically invoked together with a File Compare (see 7.1), e.g. by **Query|Show Changes** from the Project Window (see 2).

The table displays all properties of both files/directories with their **Name**, **State**, **Old Value** and **New Value**. **Old Value** corresponds to the value of the first file/directory and a **New Value** corresponds to the value of the second file/directory. When the properties compare has been invoked for a versioned file or directory, *old* refers to the pristine copy of the file/directory and *new* refers to the working copy file/directory. The **State** column shows the property's state, either **Modified**, **Added**, **Removed** or **Unchanged**. The **Name** column renders the property's state by using different colors, similar to the File Compare (see 7.1).

The lower area of the dialog shows the differences between **Old Value** and **New Value** for the currently selected property, similar to the File Compare (see 7.1).

7.2.1 File menu

- Use **Close** to close the frame.

7.2.2 Edit menu

- Use **Customize** to customize accelerators (see Section 8.18).

7.2.3 Window menu

Refer to Section 2.5.12 for more details.

7.3 Compare Repository Files or Directories

The *Compare Repository Files or Directories* frame is the result of a Compare Repository Files or Directories command (see 3.10.6) invoked from the Project Window (see 2). It shows the **Directories/Files** differences between two repository directories, a *From* directory and a *To* directory.

For every file, the table shows the corresponding **Name**. The **Name** column also shows the “state” icon of the file (the same for directories). Possible states are: **Added**, **Modified**, **Modified (properties only)**, **Modified (content and properties)**, **Removed**

and **Unchanged**; they are always referring to the *modification* from *From* to *To* directory. The corresponding icons can be found in Table 2.5 and Table 2.6.

While the state displayed in the **Name** is a combination of file content and properties, the **Content** column refers only to the state of the content. The **Properties** column refers only to the state of the properties; valid states for properties are **Modified** and **Unchanged**. The **Relative Directory** displays the file's path relative to the compare directory.

7.3.1 Compare menu

- Use **Show Changes** to open a File Compare (see 7.1) which shows the differences for the currently selected file between *From* and *To* directory.
- Use **Close** to close the frame.

7.3.2 Edit menu

- Use **File Filter** to position the cursor in the file table's filter field.
- Use **Customize** to customize accelerators (see Section 8.18).

7.3.3 View menu

- Select **Files from Subdirectories** to toggle the display of files from subdirectories of the currently selected directory.

7.3.4 Window menu

Refer to Section 2.5.12 for more details.

7.4 Conflict Solver

The Conflict Solver is a kind of *Three-Way-Merge*. The content of the current file (which contains the conflicts) is displayed in the center text area (“merge view”). The left and right text areas show the contents of the two files, which have been forked from the common base. The common base itself is not displayed, but used by the UI for highlighting changes and conflicts. All file contents are taken directly from the files, which SVN produces in case of conflicting changes. The Conflict Solver is invoked by **Tools|Conflict Solver** from the Project Window (see 2).

Note	Depending on your Conflict Solver settings (see 8.6), performing a conflict solver can also invoke an external three-way-merge tool. This section only refers to the built-in <i>Conflict Solver</i> of SmartSVN.
-------------	---

The Conflict Solver works similar to the File Compare (see 7.1), see also Section 7.1.1 for details.

7.4.1 File menu

- Use **Save** to save changes to the merged file. SmartSVN will detect, whether all conflicts have been resolved and in this case also automatically mark the file as resolved (see Section 3.5.14).
- Use **Close** to close the frame.

7.4.2 Edit menu

Refer to Section 7.1.

7.4.3 View menu

- Use **All** to display all three file contents side-by-side.
- Use **Left and Merge** to display the left view beside the merge view.
- Use **Merge and Right** to display the merge view beside the right view.
- Use **Left and Right Above Merge** to display the merge view below the left and right view.
- **Ignore Whitespace for Line Comparison**, refer to Section 7.1.8.
- **Ignore Case Change for Line Comparison**, refer to Section 7.1.8.
- **Settings**, refer to Section 7.1.7. In addition, on the **Compare** page, **Compare with Base** can be selected. With this option selected, the content of the center component will not only be compared against the left and the right content, but also against the (invisible) content of the base file: Lines in the left, center and right content which are not equal are also compared to the corresponding lines of the base file and the highlighting depends on the result of this comparison.

7.4.4 Go To menu

In the addition to the **Go To** found in the File Compare (see 7.1.5), following commands are available:

- Use **Previous Conflict** to scroll to the previous conflicting difference within the active view, relative to the current cursor location.
- Use **Next Conflict** to scroll to the next conflicting difference within the active view, relative to the current cursor location.

7.4.5 Window menu

Refer to Section 2.5.12 for more details.

7.5 Revision Compare

The Revision Compare is an optimized “multi-file” compare. It gives a detailed overview of changes within a set of files. A Revision Compare is for instance invoked by **Query|Show Changes** from the Log (see 7.6) when having a revision selected. There are various other ways/windows to invoke a Revision Compare in SmartSVN.

The core component of the Revision Compare is a read-only File Compare (see 7.1) view; for details regarding the usage, refer to Section 7.1.1. The upper part of the Revision Compare frame consists of a directory tree and a file table, which displays the files being part of the Revision Compare.

7.5.1 File menu

- Use **Compare** to open a File Compare (see 7.1) for the selected file.

7.5.2 Edit menu

- Use **Customize** to customize accelerators (see Section 8.18).

Refer to Section 7.1 for details.

7.5.3 View menu

- Select **Files From Subdirectories** to also display files from subdirectories of the currently selected directory. This works as for the Project Window, see Section 2.4.
- Use **Refresh** to refresh the file contents and re-perform the comparison.
- **Ignore Whitespace for Line Comparison**, refer to Section 7.1.7.
- **Ignore Case Change for Line Comparison**, refer to Section 7.1.7.
- **Settings**, refer to Section 7.1.8.

7.5.4 Go To menu

Refer to Section 7.1.1 for details.

7.5.5 Window menu

Refer to Section 2.5.12 for more details.

7.6 Log

The **Log** window shows the history of a versioned file or directory (“entry”). A Log is typically invoked via **Query|Log** from the Project Window (see 2), but there are several other ways to open the Log window in SmartSVN.

The central component of the **Log** window is the **Revisions** table, which shows the found revisions together with their attributes. You can filter out certain revisions by using the **Revision Filter** field on the top right of the **Revisions** table. To the right of the **Revisions** table, the detailed **Revision Info** of the currently selected revision is displayed.

The lower part of the window shows the **Directories/Files** view for the selected revision. The displayed structure is restricted to the files and directories that are children of the *log context root*; all other files/directories which have been modified within this revision are skipped.

The *log context root* depends on the context from which the log has been invoked. For example, the log context root for logs performed by **Query|Log** from the Project Window (see 2) is either the corresponding project root directory, or the Externals (see 3.8.7) root directory. The context root can be enlarged to the corresponding Project Root (see 3.9.1) if necessary.

Note For repositories in Subversion 1.6 format, the received log data contains information on whether a changed entry is of file or directory type. Unfortunately this information is not present for older servers, hence SmartSVN tries to detect the entry types itself. The more log information is present, the better the results of this detection process. However, without complete log information SmartSVN may still be wrong. In this case, the entry is assumed to be a file (although it might actually be a directory).

If *merged* revisions are loaded via **Log|Load Merged Revisions**, they are added in a tree-like manner to their parent revision, which can then be *expanded* or *collapsed*. Because merged revisions have no direct link to the logged revisions themselves, various commands subsequently listed will not be applicable for these revisions. The context root for merged revisions is the corresponding repository root.

At all times, exactly one of the four views is “active”, which is indicated by its high-lighted tab title. Menu and toolbar actions are always performed on the currently active view.

7.6.1 Log menu

- Use **Load Properties** to fetch all properties for all displayed revisions from the repository. The **Revisions** table will be extended by corresponding table columns, one for each property. This command is only available for file Logs. The upper limit of columns to be added can be configured through the system properties (see 12.1).
- Use **Load Merged Revisions** to load the originating revisions for revisions that have been merged. This option recursively descends into merged revisions and, depending

on the number of merges that have affected the file/directory, this may cause a large or even huge number of revisions to be loaded.

- Use **Export to File** to export the log information to a file. Refer to Section [7.6.7](#) for details.
- Use **Close** to close the frame.

7.6.2 Edit menu

- Use **Stop** to cancel the currently running operation.
- Use **Open** to open the selected revision/file/directory. For details, refer to Section [2.5.2](#). This command will only be applicable for revisions of file Logs.
- Use **Copy Message** to copy the commit message of the selected revision to the clipboard.
- Use **Copy Name** to copy the name of the selected file to the clipboard. If multiple files are selected, all names will be copied, each on a new line.
- Use **Copy Path** to copy the path of the selected file relative to the log context root to the clipboard. If multiple files are selected, all paths will be copied, each on a new line.
- Use **Customize** to customize accelerators (see Section [8.18](#)).

7.6.3 View menu

- Select **Skip Unchanged Revisions** to skip revisions for which the logged entry has not actually been changed, but has only been reported due to a copy operation of one of its parents. For example, when creating a Tag (see [3.9](#)) of the project root, the log for every entry of that tag will contain this tag-revision.
- Select **Show Files Recursively** to toggle recursive file listing in the **Files** view. If the file listing is recursive, the **Files** view will display not only all files in the directory selected in the **Directories** view, but also all files in subdirectories of the selected directory.
- Select **Show Only Entries Below Selected Directory** to restrict the **Directories/Files** view to only those directories and files which are actually children of the logged directory.

7.6.4 Modify menu

- Use **Change Commit Message** to change the commit message of the currently selected revision. Enter the new **Commit Message** and wait, if necessary, until SmartSVN has finished rebuilding the corresponding Log Cache (see [5.3](#)).

- Use **Merge Revision** to merge the selected revision/file/directory to your local working copy. If you want to configure advanced options for the merge, use the default Merge command (see 3.7.1).
- Use **Rollback Revision** to roll back the selected revision/file/directory locally, i.e. in your local working copy. You may then review the rolled back changes and, if acceptable, commit them (see 3.6). This command will only be applicable for logs which are linked to a local working copy.

7.6.5 Query menu

- Use **Show Changes** to compare the selected revision/file/directory against its preceding revision or to compare two selected revisions/files/directories against each other. Depending on whether two files or directories are compared, either the File Compare (see 7.1) or the Properties Compare (see 7.2) will come up. When invoking **Show Changes** on a revision, the Revision Compare (see 7.5) will come up.
- Use **Compare with Working Copy** to compare the selected revision/file against the file's working copy within your project. This command is only applicable to revisions of file Logs.
- Use **Log** to perform another Log for the selected file/directory. This command is not applicable to revisions as it would result in a Log that is indential to the present one.
- Use **Revision Graph** to create a Revision Graph (see 7.7) for the selected revision/file/directory.
- Use **Annotate** to Annotate (see 7.8) the selected revision/file. This command is only applicable to revisions of file Logs.
- Use **Save As** to save the contents of the selected revision/file to a local file, for details refer to (see 4.5). This command is only applicable to revisions of file Logs.

7.6.6 Window menu

Refer to Section 2.5.12 for more details.

7.6.7 File Export

You can export log data in various formats to a file using **Log|Export to File**.

Select either to export **All revisions**, independent of the selection or only the **Selected revisions**. Specify the **Output File** to which the log information will be written. If **Include changed paths** is selected, not only the main revision information but also the details on which files/directories have been changed will be exported.

Specify the file **Format** which shall be used for the export. **XML** will export in raw XML format, as used by `svn log --xml`. **HTML** will give a basic *HTML* output. **Plain**

text will give a simply formatted plain text file. **Custom** maybe used to export in an arbitrary format, by performing a style sheet transformation on the raw XML data. In this case, enter the path of the stylesheet for **XSLT-File**.

7.7 Revision Graph

The **Revision Graph** is a standalone window that allows you to browse the history of a repository, which can be either the history of the entire repository, or the history of some directory or file inside the repository.

There are several ways to open the Revision Graph in SmartSVN, one of them being through the menu of the Project Window (see 2) via **Query|Revision Graph**.

The Revision Graph consists of several views: **Revisions**, **Revision Info**, **Directories** and **Files**. The Revisions view is basically a table of revisions with a graph showing the parent-child relationships between the revisions. This will be discussed in more detail in Section 7.7.1.

The three other views display additional information related to the revision currently selected in the Revisions view:

- The **Revision Info** view displays various attributes of the selected revision, such as commit message, revision number, commit date, author, etc.
- The **Directories** and **Files** view provide a file-manager-like view of the files that were modified as part of the selected revision. What is shown in the Files view depends on the selection in the Directories view; that is, it shows all modified files in the revision beneath the directory currently selected in the Directories view.

At any time, exactly one of the four aforementioned views is “active”, as indicated by the color of the tab titles. Menu and toolbar actions are always performed on the currently active view.

Note As the Revision Graph displays branches and tags, the Tag-Branch-Layout (see 3.9.1) must be configured properly.

7.7.1 Revisions view

The Revisions view lists all revisions in the repository sorted by date, with the newest revisions on top.

Each of the various table columns displays a certain revision attribute, such as **Message** and **Revision**. You can hide and unhide columns by right-clicking on the column header area and selecting or deselecting entries in the context-menu. You can also reorder the columns via drag and drop of the column headers.

The **Message** column contains a graph showing the parent-child relationships between the revisions. Generally, the graph will consist of vertical lines of different color, each of which represents a branch of development. These branches may split or merge to represent branching and merging operations that occurred at some point in the repository history. There are four main types of relationships, expressed by different line styles:

- **Normal parent-child relationship:** this is the default relationship when performing a new commit. It's displayed by thick, colored lines.
- **Complete merge relationship:** it is set up by performing a merge commit for which all source revisions have been merged into the target. It's displayed by thin, colored lines when having *Merge Arrows* loaded by **Query|Show Merge Arrows**.
- **Partial merge relationship:** is set up by performing a partial merge (cherry-pick) for which not all source revisions have been merged into the target. It's displayed by thin, colored, dashed lines when having *Merge Arrows* loaded.
- **URL relationship:** exists between branches which have the same URL, but are not related. This is typically the case, when removing and re-adding a branch. It's displayed by thin, gray lines when having **View|Join Same Locations** selected.

At the end of each branch, you will find a revision that has a colored marker attached to it, i.e. a colored box containing the name of the branch the revision belongs to, such as “trunk” or “feature-branch”. These markers will of course move along as you commit into the respective branches. In addition to the branch markers, there are also markers for tags, labelled with the names of your tags, e.g. “1.0” or “1.5-beta-1”.

On the top right of the Revisions view is a **Filter** text field that allows you to enter expressions for filtering out some of the revisions in the Revision Graph. This filter works just like the one on the file table (see 2.4.2). Additionally, the drop-down menu on the left of the filter field allows you to include or exclude certain revision attributes from being matched against, namely Author, Branch and Message. For example, if only the Author field is selected, the filter expression entered into the filter field will only be matched against the author fields of the revisions.

To the right of the filter field is a drop-down menu for selectively including or excluding branches from being displayed in the Revision Graph. With this, you can for instance configure the Revision Graph to only display the branches “branch1” and “branch2”.

7.7.2 Merge Information

The Revision Graph can display information on which revisions have been merged from other revisions in various ways. Depending on the selected visualization method, it may be necessary to fetch SVN's *mergeinfo* for every displayed revision from the repository, what may take a while. SmartSVN will cache this *mergeinfo* for the current graph, so subsequent invocations of mergeinfo-related queries are performed much faster.

Merge Coloring

By default, lines are colored by chunks, where coloring changes at forks. When using **View|Merge Coloring**, SmartSVN will for the currently selected revision (“target”) classify (and color) every other revision into one of the following categories:

- **Merged Now:** The revision has been merged directly at and to the selected target (by default displayed using light green).

- **Merged:** The revision has been merged into the selected target, but not at the target's revision itself (by default displayed using dark green).
- **Not Yet Merged:** The revision has not yet been merged into the selected target (by default displayed using light red).
- **Not mergable (normal revision):** The revision is in the ancestor line of the selected target and hence can't be merged (displayed using black).
- **Unknown:** used when no merge information is present for the selected target (displayed using gray).

The colors can be configured in the Revision Graph settings (see [8.9](#)).

Warning! The merge coloring depends on which merge information is currently loaded. To make sure merge information for the currently selected revision is present, use **QueryShow Merge Sources**.

7.7.3 Graph menu

- Use **Select Shown Branches** to configure which branches to display in the graph.
- Use **Export as HTML** to export the complete graph to an HTML file.
- Use **Close** to close the frame.

7.7.4 Edit menu

Refer to Section [7.6.2](#) for more details.

7.7.5 View menu

- Use **Branch Coloring** to switch to the default graph coloring.
- Use **Merge Coloring** to switch to Merge Coloring (see [7.7.2](#)).
- Select **Show Dead Tags and Branches** to toggle the display of tags and branches which are not present anymore in the repository's *HEAD* revision.
- Select **Join Same Locations** to display revisions having the same URL in the same branch (column). For details refer to Section [7.7.1](#).

Refer to Section [7.6.2](#) for more details.

7.7.6 Modify menu

Refer to Section [7.6.4](#) for details.

7.7.7 Query menu

- Use **Show Merge Arrows** to add *merge arrows* to the graph. Refer to Section [7.7.1](#) for details.
- Use **Query|Show Merge Sources** to display which revisions have been merged into the currently selected *target* revision. Refer to Section [7.7.2](#) for details.
- Use **Clear Merge Information** to clear the currently displayed merge information (and the cached revision mergeinfo).

Refer to Section [7.6.5](#) for details.

7.7.8 Window menu

Refer to Section [2.5.12](#) for more details.

7.8 Annotate

The **Annotate** window shows the contents of a file with each line prefixed by the line number and by information about the *last* revision in which this line has been introduced or changed. The **Annotate** window is typically opened via **Query|Annotate** from the Project Window (see [2](#)), but there are other ways and windows to open an **Annotate** window in SmartSVN.

The **Revision** drop-down list displays all available revisions of the file, and allows navigating through them.

With the **Color By** drop-down list you can select one of the following line coloring schemes:

- Choose **Revision** to have two colors and a threshold revision **Newer Or Equal**. Lines which have been introduced before this threshold revision will receive the default background color, while lines introduced at or after the threshold revision will receive another background color.
- Choose **Age** to change the coloring scheme so that the colors of the lines reflect their “Age”: The youngest and oldest line will be determined, receiving two distinct colors. For all other lines, the color will be linearly interpolated based on their relative age compared to the youngest and oldest line. The interpolation itself can either be based on the **Revision** number or on the revision’s commit **Time**.
- Choose **Author** to have lines of the same author displayed with the same background color, and lines of different authors displayed with different background colors.

7.8.1 Annotate menu

- Use **Close** to close the frame.

7.8.2 Edit menu

The **Edit** menu contains common functions to alter the file contents and to find a certain text sequence within the file contents.

- Use **Customize** to customize accelerators (see Section 8.18).

7.8.3 View menu

- **Settings**, refer to Section 7.1.7.

7.8.4 Revision menu

- Use **Show File Changes** to invoke a File Compare (see 7.1) between the currently selected **Revision** and the previous revision.
- Use **Show Revision Changes** to invoke a Revision Compare (see 7.5) containing all changed files between the currently selected **Revision** and the previous revision.
- Use **Go To First Revision** to select the first **Revision**.
- Use **Go To Last Revision** to select the last **Revision**.
- Use **Go To Next Revision** to select the next **Revision**.
- Use **Go To Previous Revision** to select the previous **Revision**.
- Use **Go To Preceding Revision** to select the preceding **Revision** for the currently selected line – to see what the content of the line has been before.

7.8.5 Go To menu

- Use **Go to Line** to jump to a certain line in the file contents.

7.8.6 Window menu

- Use **Full Screen** to switch to full screen mode. Select this menu item again to go from full screen mode back to normal.
- Use **Minimize** to minimize the Annotate window.
- At the bottom of this menu, there will be additional menu entries to switch between SmartSVN windows.

7.9 Merge Preview

The *Merge Preview* is the result of a Merge command (see 3.7.1) invoked from the Project Window (see 2). It shows a **Directories/Files** structure of which files and directories will be affected by the merge.

For every file, the table shows the corresponding **Name** and its **Relative Directory**, according to the merge root. **State** shows the merge state for the file, either *Modified*, *Added*, *Removed*, *Unchanged* or *Skipped* For *Modified* files, both the **Content** and the **Properties** can be either *Conflicting*, *Modified* or *Unchanged*. *Skipped* files can't be processed by the merge, e.g. because they have been renamed or moved in the merge source, i.e. the local working copy.

7.9.1 Merge menu

- Use **Show Changes** to show the File Compare (see 7.1) between the current *local* file and the merge *Result* for the selected file. This command will only be applicable for *Modified* and for *Conflicting* files.
- Use **Show 3-Way-Merge Changes** to show the Conflict Solver (see 7.4) for the selected file, previewing the detailed changes and conflicts which can be expected when actually performing the merge.
- Use **Perform Merge** to actually perform the merge exactly as it has been previewed here. If you had initially selected a merge revision range containing *HEAD*, these ranges will have been adjusted. This prevents the final merge from including any new revisions which had been committed after previewing the merge.
- Use **Close** to close the frame.

7.9.2 Edit menu

- Use **Customize** to customize accelerators (see Section 8.18).

7.9.3 View menu

- Select **Files From Subdirectories** to toggle the display of files from subdirectories of the currently selected directory.

7.9.4 Window menu

Refer to Section 2.5.12 for more details.

Chapter 8

Preferences

The application preferences define the global behavior of SmartSVN, regarding UI, SVN commands, etc. Contrary to the project settings (see Section 6.3), these preferences apply to all projects.

Tip	Most preferences are stored in the <code>settings.xml</code> file in SmartSVN's settings directory. Refer to Section 11 for details.
------------	--

8.1 On Start-Up

These settings configure the startup behavior of SmartSVN.

You can either choose to **Open last project**, **Show Welcome Dialog** or **Do nothing**, i.e. start with an empty main frame.

Select **Remove obsolete projects** to check for every project on start-up whether the corresponding root directory still exists. In case the root paths of certain projects are not valid anymore, you will be asked whether these projects should be removed from the project tree (see Section 6.2).

8.2 Project

For **Open Project** you can specify the behavior when opening a project. Projects can be opened **In current window** (unless there are SVN operations active for the currently opened project) or **In new window**. By default, **Ask** is selected to let you choose individually.

With **Confirm closing** selected, you will always be asked before a project is closed.

8.3 Authentication

The **Authentication** page contains all the settings needed for establishing connections to repositories. This includes authentication information, such as user names and passwords.

When opening a working copy, SmartSVN automatically creates a so-called *repository profile* for the working copy, if no such profile exists yet. A repository profile is a collection

of all the information needed to connect to the repository associated with the working copy. The profile is incomplete in the beginning and will be filled out as needed. For instance, as soon as you specify a user name and password, or a certificate, to perform operations that require access to the repository, the information you provided is stored in the profile.

The table on the **Credentials** tab of the **Authentication** page lists all existing repository profiles and their associated authentication information. Use the **Edit** and **Delete** buttons to edit and delete profiles, respectively. Editing profiles is described in more detail in Section 8.3.1.

Use the **Show Passwords** button to display the passwords in the **Password/Passphrase** column in plain text. The **Change Master Password** button allows you to set, reset and change the master password used for encrypting all stored passwords. For more information on passwords refer to Section 8.3.4.

The **Proxies** and **Tunnels** tabs allow you to configure the proxies and tunnels through which SmartSVN may connect to repositories. See Section 8.3.2 and Section 8.3.3 for more information on proxies and tunnels, respectively.

8.3.1 Editing Profiles

On the **Credentials** tab of the **Authentication** page in the preferences, you can click on the **Edit** button to edit the currently selected profile. Clicking the button opens a dialog with various options (namely **Credentials**, **Client Certificate** and **Proxy**), which will be discussed in the following subsections.

At the top of the dialog, you can specify an **Optional Name** for the repository profile. This name is not only used for increased readability when the profile is displayed; you can also enter this name directly in many fields across SmartSVN wherever an URL is expected. For instance, if you do a checkout (see Section 3.1), you can enter this name instead of the full URL in order to specify the repository to check out from.

At the bottom of the dialog, there's a **Verify connection** checkbox. If this checkbox is checked, SmartSVN will try to connect to the repository with the specified connection information when the **OK** button is pressed, and inform you of any failures (which may be due to an incorrect configuration).

Credentials

On the **Credentials** tab of the **Edit Repository Profile** dialog, specify the type of **SVN Login**, which can be either **Anonymous** or **User Name and Password**. With the latter option, you have to supply a **User Name** and an optional **Password**. If a password was specified, mark the **Save password** checkbox so that the password is stored. See Section 8.3.4 for more information on password storage.

Client Certificate

If applicable, the **Edit Repository Profile** dialog will display a **Client Certificate** tab. On this tab, you can enable or disable client authentication for the edited profile. If enabled,

a **Certificate File** and the corresponding **Certificate Passphrase** must be specified. If you enter a passphrase, mark the **Save passphrase** checkbox so it will be stored.

Proxy

On the **Proxy** tab, you may optionally specify a proxy through which to connect to the repository.

You can add proxies to the drop-down list of available proxies by clicking on the **Add** button. This will open a dialog where you can enter the **Host** and **Port** of the proxy to be added.

If a proxy is selected, you can configure the **Proxy Login**: Either **Anonymous** or with a **User Name** and **Password**. In the latter case, enter a user name and an optional password in the fields below. If you enter a password, mark the **Save password** checkbox to have the password stored.

Note that the login information entered on the **Proxy** tab is "global" in the sense that it is not stored on a per-profile basis. Rather, the login information entered here will affect all profiles that use this proxy.

For each *proxy configuration* you have to specify the configuration's **Name** and the proxy **Host** and **Port**. For **Login**, select either **Anonymous** if the proxy itself requires no authentication or **User Name and Password**. In the second case, specify the required **User Name** and **Password**. You can choose to **Save password**, see also Section 8.3.4.

8.3.2 Proxies

On the **Proxies** tab, you can specify the proxy hosts to be used to connect to SVN repositories via the *HTTP/HTTPS* protocol. After being added, a proxy can be used within repository profiles.

Use the **Add** and **Edit** buttons to add or edit proxies, respectively. Both actions require entering a proxy **Host** and **Port**. Use the **Delete** button to delete the selected proxy.

8.3.3 Tunnels

On the **Tunnels** tab, you can configure custom *svn+ssh* tunnels. Tunnels are useful when you already have a working SSH infrastructure that handles authentication and communication. The configured tunnels can then be used within repository profiles.

A tunnel has a **Name**, a tunnel **Command** and **Parameters** for the tunnel command. The **Command** typically is an *ssh* executable, like PuTTY's `plink.exe` on Microsoft Windows or `ssh` on Unix and Mac OS X. The tunnel command is always invoked when a *svn+ssh* connection is set up and handles the complete SSH-based communication between SmartSVN and the server. The **Parameters** can contain predefined variables which are filled out with concrete values from the corresponding repository profile when the tunnel is used:

- **Host**: The host name of the server
- **Port**: The port number on the server

- **SSH Login Name:** The login name on the server
- **'svnserve' Start Command:** The command to start the `svnserve` process. Either this variable or the actual start command must show up in the **Parameters** definition.

8.3.4 Passwords

All passwords needed to access repositories can optionally be stored in SmartSVN's password store. This password store is located in the `password` file, which can be found in SmartSVN's settings directory (see Section 11).

The password store can be protected by a *master password*, and each time you start SmartSVN, this master password has to be entered as soon as SmartSVN tries to access the password store for the first time. The entered password is kept in memory while the program runs, so you don't have to enter it again for the rest of the current session.

You may choose **Don't use a master password** if you don't want the password store to be protected with a master password. However, this option is only recommended if you can make sure the master password file itself is protected against unauthorized access.

By clicking on the **Change Master Password** button on the **Authentication** page in the preferences, you can set, reset or change the master password.

Use **Change master password** to *change* the current password; this will preserve the stored passwords, but requires that you supply the **Current Master Password**. Note that you don't need to enter the **Current Master Password** if you are currently working without a master password.

If you have forgotten the master password, select **Set new master password**. In that case all previously stored passwords will be discarded.

Enter the **New Master Password** and **Retype New Master Password**. When leaving both fields blank, you will continue to work without a master password, i.e. as if having selected the option **Don't use a master password** when you were asked to set the master password.

8.4 User Interface

These settings affect various aspects of SmartSVN's user interface.

You can select whether to use **Basic** or **Advanced** recursion options. For details, refer to Section 3.14.1.

Select **Hide ignored and repository-only directories according to View-menu filters** to apply the filters from the **View** menu within the Project Window (see 2) not only to files, but also to ignored and repository-only directories. For more information on the **View** menu, refer to Section 2.5.3.

With the option **Use background color in file table to indicate certain states** you can control whether the file table on the Project Window (see 2) may change its background color to indicate certain table states. For example, the table may take on a yellow background to indicate it's currently showing files that match the filter pattern entered in the filter text field on the top right of the file table.

Select **Show file and directory tooltips** to toggle the display of tooltips for the **Directories** tree and the **Files** table within the Project Window (see 2).

For **File Name Matches** you can configure the filename search and filter features in SmartSVN:

- **Exact case:** Requires the search pattern and file name to match in case.
- **Ignore case:** Ignores the case for matching search pattern and file name.
- **Smart upper case:** Lower case characters in the search pattern can match upper- and lower-case characters in the file name. But upper-case characters in the search pattern match only upper-case characters in the file name. Examples: **SMF** will match `SuMainFrame`, but not `SuMainContentFrame`. **fileS** will match `FileSignature`, but not `Files`.

Select **Nest in System Tray** to have SmartSVN show a *System Tray* icon. This option is not available for all operating systems. For details refer to Section 9.8.

Configure the **Date Format** and **Time Format** to be used by SmartSVN when displaying dates and times, respectively, and combinations of both. These formats have no effect on SVN operations. It's recommended to restart the application after having changed these formats.

8.5 Commit

Here you can configure global commit (see 3.6) options.

Select **Show directory configuration page** if the commit wizard should display the configuration page with directory-related options, e.g. subdirectory recursion options.

Select **Skip Change Set entries** to ignore changed files or directories which have already been assigned to a Change Set (see 3.13).

Select **Detect moved and renamed files** if you want SmartSVN to detect files which are most likely renamed or moved. These files will not simply be added and removed, but marked as copied. For details, refer to Section 3.5.8.

Except from those files which have been selected and which are in a committable SVN state, SmartSVN can **Suggest To** commit further files: Select **Add unversioned files and directories** to also report unversioned (most likely new) files and directories. Select **Remove missing files and directories** to also report missing (most likely obsolete) files and directories.

Select **Remove removed parent directories** to make SmartSVN also scan parent directories of the files/directory which have been selected for the commit. If such a parent directory is scheduled for removal, it will also be suggested for the commit. With **Also remove empty parent directories**, all resulting empty parent directories will also be suggested for the commit.

Tip	To clean up all empty directories within your project, you can use Tools Remove Empty Directories , see Section 10.3.
------------	--

Select **Remind me to enter a commit message** to make SmartSVN warn you when trying to commit without a message. Select **Trim whitespaces from commit message** to trim leading and trailing whitespaces from the commit message directly before committing.

Specify to **Remember up to** a specific amount of **entered commit messages** for each project.

Choose **For File Commits** if you want to be warned for potentially missed files when performing a commit:

- Select **Do not warn for potentially missed files or directories** to switch all warnings off.
- Select **Warn for potentially missed directories, just up the root** to receive a warning if you have selected *all visible committable* files and any of their parent directories is modified (containing properties changes).
- Select **Warn for any potentially missed directories** to receive a warning if you have selected *all visible committable* files and there are any more modified directories in the project.
- Select **Warn for any potentially missed directories and files** to receive a warning if you have selected *all visible committable* files and there are any more modified directories or committable files.

8.6 Conflict Solver

Here you can configure external tools which should be used instead of the built-in Conflict Solver (see 7.4).

You can either choose to use the **Built-in Conflict Solver** or an **External Conflict Solver**. An external conflict solver is defined using the operating system **Command** to be executed, along with its **Arguments**.

Arguments are passed to the **Command** as it would occur from the OS command line. The place holders `${leftFile}`, `${rightFile}`, `${mergedFile}` and `${baseFile}` can be used, which will be substituted by the absolute file path of the left, right and resulting merged file, respectively. Furthermore, the place holder `${encoding}` can be used, which will be substituted by the encoding used for the file. Refer to Section 6.3.1 for details.

8.7 Open

With **Don't open or compare more than X files at once**, you can specify an upper limit beyond which you will be asked before a certain set of files is opened all at once. It is recommended not to set this value too high, to prevent accidentally opening a large number of files.

8.8 Refresh

With these settings you can configure the refreshing of the file system.

Choose **Recursively scan unversioned directories** to make SmartSVN descend into unversioned directories and display the complete unversioned sub-tree. Otherwise, only the unversioned root directory itself will be scanned and displayed.

With **Manually Refresh** you can configure how the manual Refresh via **View|Refresh** (see Section 2.4) behaves. All options take into account the scanned/unscanned state of the working copy, see Section 6.3.2.

- You have the option to refresh **Always root directory**. In this case the directory selection in the tree does not matter; the whole project is always refreshed. This option is the most expensive in terms of system resources, but will guarantee that after changing the selection in the tree, displayed data is up to date (relative to the last refresh time).
- You can also choose to refresh only the **Selected directory recursively**. This option can be useful if you know you are only working in a specific part of your SVN project.
- The option **Selected directory (recursively if set for view)** also refreshes only the selected directory. Whether this refresh is recursive or not depends on **View|Files From Subdirectories**. This option is the fastest way of refreshing as it is most selective, but it requires you to always be aware of which directories you have refreshed and hence which information displayed in the directory tree, and file table, is actually up to date.

To automatically perform a Remote State Refresh with every local Refresh, you can select **Refresh Remote State with local Refresh**. You may choose to **Include externals** and you may choose to **Scan locks** for a remote state refresh. For details regarding the *Remote State*, refer to Section 3.12.

8.9 Revision Graph

Here you can configure global Revision Graph (see 7.7) options.

If the option **Show dialog to allow root selection** is selected, opening the Revision Graph will initially show a dialog with various options, e.g. the root directory for which the Revision Graph will be created.

The **Colors** are used to colorize the **Branches** and **Revisions** of a Revision Graph. You can specify colors for both **Normal** (unselected) and **Selected** mode. Use **Reset to Defaults** to reset the colors to SmartSVN's default values.

8.10 Built-in Text Editors

These settings are used as a default for all text-displaying and editing views of SmartSVN, like the File Compare (see 7.1), the Conflict Solver (see 7.4), the Annotate (see 7.8) and

the Changes view (see 2.6).

For the **Font** page, choose the **Font Family** and the **Font Size** to be used by SmartSVN's text components.

On the **Colors** page you can customize the various colors used by SmartSVN's text components. Use **Reset to Defaults** to restore the default settings for **Colors** page.

On the **Behavior** page, you can configure various text editing features.

8.11 File Compare

Here you can configure external file compare tools which can be used instead of the built-in File Compare (see 7.1).

You can associate a specific **File Pattern** with a file comparator. You can either choose to use the **Built-in text file comparator**, an **External comparator** or an **External viewer**.

8.11.1 External Comparators

An external comparator is defined by the operating system **Command** to be executed, along with its **Arguments**. **Arguments** are passed to the **Command** as they would be passed from the OS command line. The optional place holders `${leftFile}` and `${rightFile}` will be substituted by the absolute file path of the left and right file to compare, respectively. In cases where SVN-internal files like the *pristine copy* is used for comparison, the content of this file is copied to a temporary location and this temporary file is passed as a parameter. The optional place holders `${leftTitle}` and `${rightTitle}` will be substituted by the left and right file title, respectively, which SmartSVN would use when displaying the file contents with its internal file comparator.

Furthermore, the place holders `${leftEncoding}` and `${rightEncoding}` will, if used, be substituted by the encoding of the left and right file, respectively. Refer to Section 6.3.1 for details.

With **In case of svn:mime-type is binary, try to detect whether actually text type** you can override binary `svn:mime-types`. In this case, SmartSVN will detect the content type *text/binary* itself by inspecting the file. This is the same as if `svn:mime-type` has not been set at all.

8.11.2 External Viewers

An external viewer is defined by the operating system **Command** to be executed, and its **Arguments**. It's executed two times, once for the left and once for the right file to "compare". **Arguments** are passed to the **Command** as they would be passed from the OS command line. The optional place holders `${file}` will be substituted by the absolute file path of the left and right file to view, respectively.

8.12 External Tools

These settings configure external tools, which can be invoked via **Edit|Open**.

You can link a specific **File Pattern** to an external tool. An external tool is defined by the operating system **Command** to be executed, along with its **Arguments** and **Run In**. **Arguments** are passed to the **Command** as it would occur from the OS command line. Additionally the place holder `${filePath}` can be used, which is substituted by the absolute file path of the file (from the file table), on which the command is invoked. **Run In** specifies to run the command either in **SmartSVN's working directory** or in the **File's directory**.

The **File Pattern** typically contains wild-card symbols (? and *) and may also consist of multiple patterns, separated by comma.

When running SmartSVN with *Java 6* (or above), you can also choose to invoke the **System Edit Command** or **System Open Command** instead of the self-defined command specified by **Following Application**.

Example

To configure Acrobat Reader (TM) as the default editor (viewer) for PDF-files, enter *.pdf for **File Pattern**, the path of Acrobat Reader Executable (e.g. on Microsoft Windows `acrord32.exe`) for **Command** and keep `${filePath}` for **Arguments**.

8.12.1 Directory Command

The **Edit|Open** command can also be performed on directories. For this case a **Directory Command** can be configured.

To be able to use **Edit|Open** on a directory, you have to select **Use following command to open a directory**. As for files you can configure the **Command** which shall be executed and the **Arguments** to be passed. The directory command will always be executed in the selected directory.

Example

On Microsoft Windows, to open the command shell for a selected directory, enter `cmd.exe` for **Command** and `/c start cmd.exe` for **Arguments**.

8.13 Transactions

This configuration page contains global Transactions (see 5) settings.

For **Refresh every** select the interval in **minutes** for which all active Transactions views should be refreshed.

To distinguish transactions of a project from those of additional URLs which are watched, project transactions will be labeled by a **Project Identifier**.

If the option **Ask for Master Password if required** is enabled, you will be asked for the master password as soon as SmartSVN tries to perform a Transactions refresh. If disabled, the Transactions refresh will be silently ignored until you enter the master password for the first time.

Refer to the system properties (see 12.1) for further configuration options which are seldom used.

8.14 Spell Checker

These settings configure the spell check support which is used primarily for the Commit command (see 3.6).

You can define multiple *Dictionaries*. Every dictionary has a **Name** which is used in the spell checker popup menu and a **Dictionary File**. In addition, there is also one optional **File for My Own Words** which can be extended by SmartSVN.

Note The **Dictionary File** has to be in *MySpell* format, however *Hunspell* also work. The **File for My Own Words** is a simple list of words.

Warning! Depending on the number and size of the dictionary files, the memory consumption of SmartSVN can increase significantly.

If you have configured multiple dictionaries, text fields for which spell-checking is supported offer a **Language** pop up menu from which you can select the intended dictionary by its **Name**. Alternatively, you can choose whether to **Use Best Matching** or **Use All** dictionaries. **Use All** is useful to combine multiple dictionaries of the same language, e.g. one file with general expressions and one with domain-specific expressions. **Use Best Matching** is useful to build a super-dictionary containing multiple languages and have SmartSVN detect which dictionary fits better for a given text.

Example

When you are frequently writing *English* as well as *German* commit messages, you can specify one English and one German dictionary and select **Use Best Matching**. Now, when writing an English commit message, SmartSVN will detect after a few words that the English dictionary fits better and hence will check the complete commit message only with the English dictionary (as if you had manually selected the English dictionary).

On the other hand, when writing a German commit message, SmartSVN will detect to use the German dictionary and only check for German spelling correctness.

8.15 Shell Integration (Windows)

These settings configure the Shell Integration (see 9.6) of SmartSVN.

Select for which drive types and in which range of functions the shell integration shall be applicable. For every drive type you can choose whether to show **Icon Overlays** (and the context menu) or only the **Context Menu** or have the shell integration be completely **Disabled**.

If necessary, specify further **Paths** for which the shell integration will only be applicable with a limited range of functions, i.e. only the **Context Menu** or completely **Disabled**. Use only plain paths, like `c:\temp` or `n:`, but no patterns here.

Note In general it's recommended to have **Icon Overlays** enabled only for **Fixed Drives**, because the display of the overlays may slow down your machine due to access to the *working copy metadata* (.svn directory).
When working copies are located on fast network shares, **Icon Overlays** should work well. In case you have a mix of fast network shares and, for example, slow VPN-tunneled shares, you can exclude the latter ones by the **Paths** input field.

8.15.1 Status Cache

Use **Configure Status Cache** to configure the Status Cache (see 9.9). This requires the *Status Cache service* to be running.

In the dialog you can configure the **Cache Roots** which will be served by the Status Cache. Enter every root directory on a new line, wildcards are *not allowed* here.

Optionally you can reset the Status Cache by **Clear all cached status information**. Selecting this option is only recommended if you definitely want to get rid of cached status information for a certain root directory as cached information is not discarded by simply removing this root directory from the **Cache Roots** list.

8.16 Shell Integration (Mac OS)

These settings configure the Shell Integration (see 9.7) integration of SmartSVN.

Select whether to enable the shell integration by **Integrate in Finder**. If necessary, specify further **Paths** for which the shell integration shall be completely disabled. Use only plain paths, like /Volumes, but no patterns here.

8.17 Check for New Version

These settings configure the *New Version Check* mechanism of SmartSVN (Section 2.5.13).

Select **Automatically check for new program version** to make SmartSVN check for program updates after it has been started. Choose either **Daily**, **Weekly** or **Monthly**; the recommended option is **Weekly**.

Note For beta versions the interval is fixed to **Daily**.

The version check reads a small file from <http://www.wandisco.com>. If necessary, you can specify a proxy server by **Use a proxy server to connect to the internet**. In this case specify **Host** and **Port** for the proxy server and optionally **Username** and **Password** to access the proxy server.

8.18 Customize

For every frame in SmartSVN you can configure accelerators, and sometimes, if applicable, also the tool bar and context menus. Use **Edit|Customize** to open the configuration dialog.

8.18.1 Toolbar (not always available)

Use this page to customize the toolbar.

Use **Add** to add one or more **Available** buttons to the toolbar, and **Remove** to remove one or more **Selected** buttons from the toolbar. From the **Add** drop down, use **Fixed Separator** to add a separator before the currently selected button. Use **Stretching Separator** to add a stretching space before the currently selected button. The remaining horizontal space is subdivided and assigned to the stretching separators. Use **Move Up** and **Move Down** to re-arrange the order of the buttons.

Select or deselect **Show text below icon** to show or hide the toolbar button text.

Note All operations can be performed via Drag-and-Drop, too.

8.18.2 Accelerators

Use this page to customize the accelerators (shortcuts).

To set or change an accelerator, select the corresponding menu item, go to the **Accelerator** field, press the key combination and click **Assign**. To remove existing accelerators, select the corresponding menu items and click **Clear**. To reset accelerators to their default, select the corresponding menu items and click **Reset**.

Tip You can double click a menu item to directly jump to the **Accelerator** field. You can assign/change multiple accelerators at the same time, if they each belong to a different **Window**.

8.18.3 Context Menus (not always available)

Use this page to customize the context menus.

First select the **Context Menu** to change. Then you will find all available menu items on the left and the current context menu structure on the right. You can either use Drag-and-Drop to arrange the context menu or use the corresponding buttons: Use the **Add** button to add a selected menu item from the left side before the selected item on the right side. You also can use **Add Separator** or **Add Menu** to add the corresponding item before the selected item on the right side. Each (sub)menu contains a gray placeholder at the end to allow adding items to the end of that (sub)menu. Use the **Remove** button to remove a selected menu item, a separator or a submenu on the right side. Use **Reset to Defaults** to restore the default context menu layout for the selected **Context Menu**.

Tip If you haven't changed the context menus (significantly) it's recommended to use **Reset to Defaults** after having upgraded SmartSVN to a new *major* version as new menu entries might have been added.

Chapter 9

Shell Integration

SmartSVN offers a *shell integration* to have the SVN functionality of SmartSVN also present in certain parts of the *GUI* shells, like in *file dialogs*. The shell integration is currently present on *Microsoft Windows* and *Apple Mac OS X*. It is only available when SmartSVN is running (except the one on Mac OS X 10.6).

9.1 Commands (Windows and OS X 10.5)

For locally versioned files and directories, the most important SVN commands are available from the shell's context menu. Performing commands from the shell's context menu results in the same dialogs and windows as if performing the commands from the Project Window (see 2). For details regarding the commands refer to Section 3.

For commands performed from the shell, the same environmental settings are used as when performing them from the Project Window. This especially implies to the Project Settings (see 6.3), if for the current working copy directory, a corresponding project exists. If no matching Project (see 6) can be found, SmartSVN will use the Default Settings (see 6.3.4).

From the context menu, use **Open Project** (or **Open SmartSVN** if no file/directory is selected) to launch the Project Window (see 2) and open the corresponding project.

Tip	For the command icons, the icon files within <code>lib/icons</code> in the installation directory of SmartSVN are used. The names correspond to the command names. For every command, there is a default icon and a <i>grayed</i> version, which has an additional <code>-g</code> in its name. If you prefer, you can replace these icons.
------------	---

9.2 Commands (OS X 10.6)

Unfortunately, Apple has dropped the Finder integration API with OS X 10.6. Hence, SmartSVN can only provide a very simple alternative using so-called services. From the Finder's context menu three commands are available if files or directories are selected: **Update from SVN**, **Commit to SVN** and **Open in SmartSVN**. Note, that because of

the limited services API these commands are available independent of the SVN state of these files or directories. They are even available for items which are not SVN-controlled. In contrast with the shell integration on Windows and OS X 10.5, SmartSVN does not need to be running to be able to invoke the commands. If necessary, SmartSVN will start automatically.

9.3 Output Window

All commands invoked from the shell integration will be executed in a special *output* window. You may select **Close automatically on success** to have the window closed automatically after all currently running operations have been completed successfully.

9.3.1 File menu

- Use **Show Changes** on a selected file/directory to see what has been changed locally by executing the command.
- Use **Log** on a selected file/directory to see the corresponding Log (see 7.6).
- Use **Close** to close the frame.

9.3.2 Edit menu

- Use **Stop** on one or more selected commands to cancel them. If no command has been selected, you will be asked whether to cancel all currently running commands.
- Use **Customize** to customize accelerators (see Section 8.18).

9.3.3 Window menu

Refer to Section 2.5.12 for more details.

9.4 Overlay Icons

The *overlay icons* show the *SVN states* for the corresponding files and directories. Currently, overlay icons are only present on *Windows*. Because the number of possible overlay icons is limited by the operating system, only the most important SVN states have a special overlay icon, see Table 9.1 for details. Versioned, but unchanged files and directories do not have a special overlay icon. For all other SVN states, the *modified* icon is used.

Tip	For the <i>overlay icons</i> , the icon files within <code>lib/icons</code> in the installation directory of SmartSVN are used. The names are corresponding to the <i>States</i> used in Table 9.1. If you prefer, you can replace these icons.
------------	---









Icon	State	Details
	Modified	File/directory is modified in contents/properties.
	Modified recursively	Directory itself or some file/subdirectory is modified (requires the Status Cache service (see 9.9) running).
	Added	File/directory is scheduled for addition.
	Removed	File/directory is scheduled for removal.
	Ignored	File/directory is not under version control (exists only locally) and is marked to be ignored.
	Conflicted	An updating command led to conflicting changes either in content or properties.
	Unversioned	File/directory is not under version control, but only exists locally.
	Root	Directory is a working root and is not modified.

Figure 9.1: Overlay Icons

The availability of overlay icons as well as commands can be configured in the Preferences (see 8.15).

Note On Windows, for technical reasons no icon overlays for files within your profile directory %USERPROFILE% are shown (except of subdirectory My Documents).

9.5 Server Mode

To provide the *shell integration* without requiring SmartSVN actually being *open*, SmartSVN can be started with the `--server-mode` argument, for details refer to Section 11.4.

9.6 Windows Shell Integration

The shell integration adds *overlay icons* to directory and file views of Windows and SVN commands in the context menu. You will especially see them for the *Windows Explorer*, but also for other software which uses the native file dialogs of Windows.

Installation

You can choose to enable the shell integration for the installation of SmartSVN, when using the *MSI* installers. It's also recommended to have SmartSVN automatically be started with the system startup, so the shell integration is available immediately. The installers offer a corresponding option which will add SmartSVN to the *Autostart* section, starting SmartSVN in server mode (see 9.5).

Uninstall

The shell integration will be uninstalled together with SmartSVN. You can also uninstall the shell integration independently from the *Control Panel, Software*, using *Repair* there.

9.7 Mac OS X Finder integration

The Finder integration lets you perform SVN commands in the Finder using the context menu.

Installation

On the first start, SmartSVN asks whether to install the Finder integration. If you choose to install it, SmartSVN will create a symbolic link `~/Library/Contextual Menu Items/SmartSVN CM.plugin`. If you choose not to install, you can install it later by selecting the option **Integrate in Finder** on the **Shell Integration** page of the Preferences (see 8.16).

If the installation by SmartSVN itself fails for some reason, you can install the Finder integration yourself. If the folder `~/Library/Contextual Menu Items` does not exist yet, create it. Right click the SmartSVN application in the Finder and select **Show Package Contents**. Copy the `SmartSVN CM.plugin` from within the SmartSVN application to the folder `~/Library/Contextual Menu Items`. Log out and login again.

Uninstallation

Unselect the option **Integrate in Finder** on the **Finder Integration** page of the **Preferences**.

To manually uninstall the Finder integration, just delete `~/Library/Contextual Menu Items/SmartSVN CM.plugin` and log out and relogin again.

Automatic start at login

The Finder integration will only work when SmartSVN is running. The easiest way to do that automatically, is to let SmartSVN be launched at login. Just right click the SmartSVN dock icon and select **Open at Login**. Alternatively, you can use the **Accounts** panel in the **System Preferences** to define SmartSVN as a Login Item. Note, that the **Hide** option has no effect. If SmartSVN is defined as a Login Item, it will be started in server mode (see 9.5).

9.8 Tray Icon

By default, SmartSVN keeps running even when all frames have been closed. To have SmartSVN still accessible, a *tray icon* is used. It's available for *Microsoft Windows*, most *Linux* desktop managers and other operating systems for which tray icons are supported.

From the context menu of the tray icon, use **New Project Window** to open a new Project Window (see 2), **New Repository Browser** to open a new Repository Browser (see 4) or **Show Transactions** to open the Transactions frame (see 5.1). Open the **Preferences** or information **About SmartSVN**. To exit SmartSVN, use **Exit SmartSVN**.

Note	On Mac OS SmartSVN is permanently available when SmartSVN is running, even when all frames are closed. In this case it has a reduced menu bar, including the Window menu.
-------------	--

The tray icon shows the progress of currently processing SVN operations which have been invoked from the shell extensions. It also shows the presence of *new* revisions for the Transactions (see 5.1) frame; the tooltip gives more information on which repositories have new transactions.

You can disable the tray icon in the Preferences (see 8.4) by deselecting **Nest in System Tray**. In this case, SmartSVN will exit once the last frame has been closed.

Note	The Nest in System Tray option is not regarded when starting SmartSVN in server mode (see 9.5).
-------------	--

9.9 Status Cache

The Status Cache is an optional *Windows service* which manages *SVN status* information for your working copies. It's primarily used to display the *recursively modified* state of directories, which shows that some files/subdirectories are modified. Also, the initial scanning/refresh (see 2.4.5) accesses Status Cache information to quickly give a preview of the working copy.

To avoid unnecessary system load, the root directories which will be served by the Status Cache have to be explicitly configured. SmartSVN will ask you to do this the first time you perform a command through the Shell Integration. The Status Cache can be reconfigured any time in the Preferences (see 8.15.1).

Performance considerations

You should carefully determine which root directories should be served by the Status Cache, as the Status Cache will introduce a certain overhead to your system's load. This overhead becomes more apparent the slower the file system to cache is. In general you should:

- Only configure to cache local harddisks
- Avoid caching of possible temporary directories which might receive temporary working copies
- Don't create a too detailed list of individual directories to cache

So for instance, if all of your working copies are located at a separate drive D:, it will be perfect to have the Status Cache configured for this single root directory D: and nothing else.

Uninstall

If you rarely work with the Shell Integration and additional *recursively modified* state is not important to you, you may completely uninstall the service. This can be done via the *Control Panel/Add or Remove Programs*, selecting the *SmartSVN* installer, **Change** and within the installer using **Change** again.

Chapter 10

Plugins

SmartSVN comes with a couple of pre-installed plugins, based on SmartSVN's *Plugin-API*. The functionality contributed by plugins may be beneficial to certain users, but is usually either not required for normal usage, or not primarily concerned with SVN.

Plugins are deployed as separate JAR files which are located in the `plugins` sub-directory in SmartSVN's installation directory. A plugin can be disabled simply by removing the corresponding JAR file from this directory.

10.1 JIRA Plugin

The *JIRA Plugin* provides a basic issue tracker integration for the *JIRA* issue tracker from *Atlassian*, see <http://www.atlassian.com/software/jira>.

The plugin adds a **Get from JIRA** entry to the drop-down menu of the commit message text fields (see Section 3.6). For the **Commit** wizard itself, it will also parse the commit message for potential JIRA issue IDs and ask whether to resolve these issues on successful commit.

10.1.1 Workflow

Before connecting to JIRA, SmartSVN will ask you for your **Username** and **Password** which may be optionally stored by **Store password**. If you are connecting to an SSL-secured JIRA server, you will have to confirm the validity of SSL-certificate fingerprints. In case SSL client authentication is required, enter the path to the **Certificate** file and its **Passphrase** which may optionally be stored by **Store passphrase**.

On the **Files** page of the **Commit** wizard, use **Get from JIRA** to display a list of JIRA issues, including their **Key**, **Summary** and **Status**. For reasons of clarity, the list will only contain issues which are assigned to your username and which are either

- in **in-progress** state or are
- contained in the next three unreleased versions (the number of unreleased versions can be changed by the system property `smartsvn.plugin.jira.unreleased-versions-to-display` for details refer to Section 12.1). If there are no unreleased versions, assigned issues for all versions will be loaded.

You can select one or more issues here which will then be set for the **Commit Message**. Using **Refresh** can be useful to reload issues from JIRA.

When proceeding the **Files** page with **Next**, the plugin will check the **Commit Message** for JIRA issue IDs. For every issue found, you will be prompted with a **Resolve JIRA Issue** dialog for which you can either select to **Mark as resolved in revision** and select the resolution revision. This will contact JIRA and resolve the issue correspondingly. **Don't mark as resolved** will leave the issue as it is.

10.1.2 Requirements

The availability of the plugin functionality for a certain working copy depends on whether bugtraq-properties (see 3.8.9) for the working copy root directory have been configured and whether the `bugtraq:url` is pointing to a JIRA *Issues* page. Following types of URLs are recognized:

- `http(s)://host:port/prefix/browse/ProjectKey-IssueID`
- `http(s)://host:port/prefix/ViewIssue.jspa?key=ProjectKey-IssueID`

The plugin only works for recent JIRA versions which provide a *SOAP interface*. The *SOAP interface* has to be enabled for your JIRA server (what can typically only be done by the administrator). For details on how to enable the interface, refer to <http://confluence.atlassian.com/display/JIRA/Creating+a+SOAP+Client>.

Note	Certain aspects of the plug-in can be customized by system properties (see 12.1).
-------------	---

10.2 Trac Plugin

The *Trac Plugin* provides a basic issue tracker integration for the *Trac* issue tracker from Edgewall Software, see <http://trac.edgewall.org>.

The plugin adds a **Get from Trac** entry to the drop-down menu of commit message text fields (see Section 3.6). For the **Commit** wizard itself, it will also parse the commit message for potential Trac ticket IDs and ask whether to resolve these tickets on successful commit.

10.2.1 Workflow

Before connecting to Trac, SmartSVN will ask you for your **Username** and **Password** which may be optionally stored by **Store password**. If you are connecting to an SSL-secured Trac server, you will have to confirm the validity of SSL-certificate fingerprints. In case SSL client authentication is required, enter the path to the **Certificate** file and its **Passphrase** which may optionally be stored by **Store passphrase**.

Warning!	Passwords and passphrases will be stored in plain-text in the <code>settings.xml</code> file (see Section 11).
-----------------	--

On the **Files** page of the **Commit** wizard, use **Get from Trac** to display a list of Trac tickets, including their **Id**, **Summary**, **Status**, **Milestone** and **Version**. For reasons of clarity, the list will only contain tickets which are assigned to your username and which are either

- in an **accepted** state or are
- contained in the next three unreleased versions respectively in the three incomplete milestones (the number of unreleased versions can be changed by the system property `smartsvn.plugin.trac.unreleased-versions-to-display`, the number of incomplete milestones by the system property `smartsvn.plugin.trac.incomplete-milestones-` for details refer to Section 12.1). If there are no unreleased versions, assigned tickets for all versions will be loaded. If there are no incomplete milestones, assigned tickets for all milestones will be loaded.

You can select one or more tickets here which will then be set for the **Commit Message**. The commit message pattern depends on the property `bugtraq:message`. If this property is not set, the pattern can be configured individually over **Configure** in the ticket list dialog. Using **Refresh** can be useful to reload tickets from Trac.

When proceeding the **Files** page with **Next**, the plugin will check the **Commit Message** for Trac ticket IDs. For every ticket found, you will be prompted with a **Resolve Trac Issue** dialog for which you can either select to **Mark as resolved in revision** and select the resolution revision and milestone (this will contact Trac and resolve the ticket correspondingly), or **Don't mark as resolved** which will leave the ticket as it is.

10.2.2 Requirements

The availability of the plugin functionality for a certain working copy depends on whether `bugtraq-properties` (see 3.8.9) for the working copy root directory have been configured and whether the `bugtraq:url` is pointing to a Trac *ticket* page. The following types of URL are recognized:

`http(s)://host:port/prefix/ticket/TicketID`

The plugin only works for recent Trac versions which provide a *Trac XML-RPC Plugin*. The *Trac XML-RPC Plugin* has to be installed and enabled for your Trac server (this is usually done by the administrator). For details on how to install and enable the Trac XML-RPC Plugin, refer to <http://trac-hacks.org/wiki/XmlRpcPlugin>.

Note Certain aspects of the plug-in can be customized by system properties (see 12.1).

10.3 Remove Empty Directories

This plugin adds the **Remove Empty Directories** menu item to the **Tools** menu. It schedules all empty, versioned directories below the currently selected directory for removal. Thereafter you can commit the selected directory to actually remove the directories from the repository.

10.4 Quick Update

This plugin adds an **Update** category to the Preferences (see 8). Here you can configure whether to **Show Update configuration dialog** or not. When there is no configuration dialog, Update (see 3.4.1) will start immediately and update the selected directory (or file) recursively to the *HEAD* revision. If you need to update to another revision, you can either enable the configuration dialog or use the Switch (see 3.4.4) command.

10.5 Plugin-API

SmartSVN's *Plugin-API* can be used to customize various aspects of SmartSVN by creating corresponding plugins. The Plugin-API currently covers following functionality:

- Modify the menu structure of the Project Window (see 2.5).
- Add custom *SVN operations* to arbitrary menus.
- Add custom file table columns (see 2.4), e.g. to show custom SVN properties.
- Customize various aspects of the Commit workflow (see 3.6).
- Customize various aspects of the Update workflow (see 3.4.1).
- Store custom Preferences (see 8) or project settings (see 6.3).

For more details refer to <http://smartsvn.wandisco.com/techarticles/pluginapi>.

10.6 Send Support Email

This plugin adds a **Contact Support** menu item to the **Help** menu which opens your email client to send a message to *smartsvn@wandisco.com*.

10.7 Hide Menu Items

You can use this plugin to remove menu items from the main menu bar of the project window (see 2). The configuration of the plugin is performed by the `menuItemsToHide.config` in SmartSVN's settings directory (see 11.1). If this file does not exist, the plugin will create it and pre-fill with all available menu items IDs, by default commented out. By un-commenting a line, the corresponding menu item will not be present anymore for the next start of SmartSVN.

10.8 Merge Info Column

This plugin adds the **Merge Info** column to the File Table (see 2.4).

10.9 Tag Multiple

This plugin adds the Tag Multiple Project Roots (see 3.9.3) functionality to the Project window (see 2).

10.10 Commit Message Templates

This plugin adds support for the `tsvn:logtemplate` property which can be used to define a default commit message which will be displayed in the Commit wizard (see 3.6). For details refer to http://tortoisesvn.net/docs/nightly/TortoiseSVN_en/ch05s15.html.

Chapter 11

Installation and Files

SmartSVN stores its configuration files per-user. The root directory of SmartSVN's configuration area contains subdirectories for every major SmartSVN version, so you can use multiple versions concurrently. The location of the configuration root directory depends on the operating system.

11.1 Location of SmartSVN's settings directory

- **Windows::** The configuration files are located below `%APPDATA%\syntevo\SmartSVN`.
Note: Before version 5, configurations files have been stored below `%USERPROFILE%\smartsvn`.
- **Mac OS::** The configuration files are located below `~/Library/Preferences/SmartSVN`.
- **Unix/Other::** The configuration files are located below `~/smartsvn`.

11.2 Notable configuration files

- `accelerators.xml` stores the accelerators (see 8.18.2) configuration.
- `license` stores your SmartSVN's *license key*.
- `log.txt` contains debug log information. It's configured via `log4j.xml`.
- `passwords` is an encrypted file and stores the passwords (see 8.3.4) used throughout SmartSVN.
- `project-defaults.xml` stores the default project settings (see 6.3.4).
- `projects.xml` stores all configured projects (see 6), including their settings.
- `repositories.xml` stores the Repository Profiles (see 8.3), except the corresponding passwords.
- `settings.xml` stores the application-wide Preferences (see 8) of SmartSVN.

- `tag-branch-layouts.xml` stores the configured Tag-Branch-Layouts (see 3.9.1).
- `transactionsFrame.xml` stores the configuration of the Transactions frame (see 5.1).
- `uiSettings.xml` stores the context menu (see 8.18.3) configuration.

11.3 Company-wide installation

For company-wide installations, the administrator can install SmartSVN on a network share. To make deployment and initial configuration for the users easier, certain configuration files can be prepared and put into the subdirectory `default` (within SmartSVN's installation directory).

When a user starts SmartSVN for the first time, the following files will be copied from the `default` directory to their private configuration area:

- `accelerators.xml`
- `project-defaults.xml`
- `repositories.xml`
- `settings.xml`
- `tag-branch-layouts.xml`
- `transactionsFrame.xml`
- `uiSettings.xml`

The `license` file (for *Enterprise* licenses and 10+ *Professional* licenses) can also be placed into the `default` directory. In this case, SmartSVN will prefill the **License** field in the **Set Up** wizard when a user starts SmartSVN for the first time. When upgrading SmartSVN, this `license` file will also be used, so users won't be prompted with a "license expired" message, and can continue working seamlessly.

Note	Typically, you will receive license files from us wrapped into a <i>ZIP</i> archive. In this case you have to unzip the contained <code>license</code> file into the <code>default</code> directory.
-------------	--

11.4 Command line arguments

SmartSVN supports a couple of command line arguments.

- `--server-mode` will just start up the core process and bring up the tray icon (see 9.8), if present. This startup mode is used for the Shell Integration (see 9).

- `--exit` will try to detect a running *SmartSVN* process and force this process to exit. This allows SmartSVN to be stopped gracefully.
- `--transactions` will bring up the Transactions Frame (see 5.1) instead of the Project Window (see 2) on startup.
- `--repository-browser` will bring up the Repository Browser (see 4) instead of the Project Window (see 2) on startup.
- `project-path` will bring up the Project Window (see 2) and load the project containing the specified *project-path*.

11.5 JRE search order (Windows)

On Windows, the `smartsvn.exe` launcher will search for an appropriate JRE in the following order (from top to bottom):

- Environment variable `SMARTSVN_JAVA_HOME`
- Sub-directory `jre` within SmartSVN's installation directory
- Environment variable `JAVA_HOME`
- Environment variable `JDK_HOME`
- Registry key `HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment`

Chapter 12

Advanced Settings

In addition to the options on the preferences dialog, SmartSVN has some advanced settings that can be set through the configuration file `smartsvn.properties`. How it is used is described in the following subsection. Additionally, you can change the program's memory limit, which is described here as well.

12.1 System Properties

SmartSVN can be configured by editing the file `smartsvn.properties` in the settings directory. The `smartsvn.properties` file contains further documentation about the available settings, so the latter will not be listed here. In this section, we will only show an example in order to give a general idea of how to alter settings in the `smartsvn.properties` file.

First, open the settings directory. Its default location is described in Section 11.1. In the settings directory, you will find the `smartsvn.properties` file. Open it with a text editor, such as Windows Notepad.

Each of the settings in `smartsvn.properties` is specified on a separate line, according to the following syntax: `key=value`

If a line starts with `#`, the entire line is treated as a comment and ignored by the program. By default, the available settings are prefixed with a `#`, so that their default values will be used. To alter a setting, uncomment it by removing the `#` character and modify the setting's value as needed.

Example

In the `smartsvn.properties` file, uncomment the following line in order to disable the splash screen:

```
#smartsvn.ui.splashscreen=false
```

12.2 Memory Limit

The memory limit (also known as maximum heap size) specifies how much RAM the SmartSVN process is allowed to use. If the set value is too low, SmartSVN may run out

of memory during memory-intensive operations. How the memory limit is set depends on your operating system:

- **Windows (all users):** In the file `bin/smartsvn.voptions` inside the SmartSVN installation directory, there is a line that looks like this: `-Xmx256m`. This sets a memory limit of 256 MB. To set a memory limit of 512 MB, change this to `-Xmx512m`.
- **Windows (current user):** The memory limit specified in `bin/smartsvn.voptions` can be overridden on a per-user basis. To do so, create a file named `voptions` in the directory `syntevo\SmartSVN` inside the application data directory. The location of the latter is usually either `C:\Documents and Settings\[Username]\Application Data` (for Windows 2000/XP) or `C:\Users\[Username]\AppData` (for Windows Vista/7). In the newly created `voptions` file, insert a line that specifies the memory limit, e.g. `-Xmx512m` for a memory limit of 512 MB.
- **Mac OS X:** Set the environment variable `SMARTSVN_MAX_HEAP_SIZE` to the desired value, e.g. `512m` for a memory limit of 512 MB. One way to set this variable for all users is to open the file `/etc/launchd.conf` with root privileges (creating it if it doesn't exist) and to add the following line: `setenv SMARTSVN_MAX_HEAP_SIZE 512m`.
- **Linux:** Set the environment variable `SMARTSVN_MAX_HEAP_SIZE` to the desired value, e.g. `512m` for a memory limit of 512 MB. One way to set this variable for all users is opening the file `/etc/profile` with root privileges and adding the following line at the end (after `unmask xxx`): `export SMARTSVN_MAX_HEAP_SIZE=512m`.