

SmartSVN 4 Reference

syntevo GmbH, www.syntevo.com

2008

Contents

1	Introduction	7
2	Project Window	8
2.1	User Interface	8
2.2	Projects	9
2.3	Directory Tree and File Table	9
2.3.1	Directory States/Directory Tree	9
2.3.2	File States/File Table	9
2.3.3	State Filters	10
2.3.4	Smart Open	10
2.3.5	Refresh	10
2.4	Menus	11
2.4.1	Project	11
2.4.2	Edit	11
2.4.3	View	12
2.4.4	Modify	13
2.4.5	Change Set (Pro Only)	14
2.4.6	Tag/Branch (Pro Only)	14
2.4.7	Query	15
2.4.8	Properties	15
2.4.9	Locks	15
2.4.10	Repository	16
2.4.11	Tools	16
2.4.12	Transactions	16
2.4.13	Window	17
2.4.14	Help	17
3	Commands	24
3.1	Check Out	24
3.2	Create Module	25
3.3	Updating	26
3.3.1	Update	27
3.3.2	Update More	27
3.3.3	Switch	27
3.3.4	Relocate	27

3.4	Local Modifications	28
3.4.1	Add	28
3.4.2	Remove	28
3.4.3	Ignore (Pro Only)	28
3.4.4	Delete Physically	29
3.4.5	Create Directory (Pro Only)	29
3.4.6	Rename	29
3.4.7	Move	29
3.4.8	Detect Moves (Pro Only)	30
3.4.9	Copy	30
3.4.10	Copy From Repository	31
3.4.11	Copy To Repository	31
3.4.12	Copy Within Repository	31
3.4.13	Revert	32
3.4.14	Mark Resolved	32
3.4.15	Mark Replaced	33
3.4.16	Clean Up	33
3.4.17	Fix (Pro Only)	33
3.5	Commit	34
3.6	Merging	37
3.6.1	Merge	38
3.6.2	Merge from 2 Sources	39
3.6.3	Reintegrate Merge	40
3.7	Properties	40
3.7.1	Edit Properties	40
3.7.2	MIME-Type	41
3.7.3	EOL-Style	41
3.7.4	Keyword Substitution	41
3.7.5	Executable-Property	42
3.7.6	Externals	42
3.7.7	Ignore Patterns	42
3.7.8	Bugtraq-Properties (Pro Only)	43
3.7.9	Merge Info	43
3.8	Tags and Branches (Pro Only)	44
3.8.1	Tag-Branch-Layout	44
3.8.2	Add Tag	45
3.8.3	Add Branch	45
3.8.4	Tag Browser	46
3.8.5	Configure Layout	46
3.9	Queries	46
3.9.1	Show Changes	46
3.9.2	Compare with Revision	46
3.9.3	Compare 2 Files (Pro Only)	47
3.9.4	Log	47
3.9.5	Revision Graph (Pro Only)	48

3.9.6	Annotate	48
3.9.7	Create Patch	49
3.9.8	Create Patch between URLs	49
3.10	Locks	49
3.10.1	Scan Repository	50
3.10.2	Lock	50
3.10.3	Unlock	50
3.10.4	Show Info	51
3.10.5	Change 'Needs Lock'	51
3.11	Remote State (Pro Only)	51
3.11.1	Refresh Remote State	52
3.11.2	Clear Remote State	52
3.12	Change Sets (Pro Only)	52
3.12.1	Move to Change Set	53
3.12.2	Move Up	53
3.12.3	Move Down	53
3.12.4	Delete	53
3.12.5	Edit Properties	54
3.13	Tools	54
3.13.1	Export Backup (Pro Only)	54
3.13.2	Conflict Solver	54
3.13.3	Repository Setup	54
3.13.4	Canonicalize URLs	55
3.13.5	Remove Empty Directories (Pro Only)	55
3.13.6	Rebuild Log Cache	55
3.14	Common Features	55
3.14.1	Recursive/Depth options	55
3.14.2	Revision input fields	56
3.14.3	Repository path input fields	56
3.14.4	Tag input fields (Pro Only)	57
4	Repository Browser	58
4.1	Repository menu	58
4.2	Edit menu	58
4.3	View	59
4.4	Modify	59
4.4.1	Create Directory	60
4.4.2	Remove	60
4.4.3	Copy/Move (Pro Only)	60
4.5	Query menu (Pro Only)	60
4.6	Window menu	61

5	Transactions	62
5.1	Transactions frame (Pro Only)	62
5.1.1	Grouping of revisions	63
5.1.2	Watched URLs	63
5.1.3	Read/Unread revisions	63
5.1.4	Display Settings	64
5.1.5	Transaction menu	64
5.1.6	Edit menu	64
5.1.7	View menu	65
5.1.8	Modify menu	65
5.1.9	Query menu	65
5.1.10	Window menu	65
5.2	Project Transactions	65
5.2.1	Settings	66
5.2.2	Transactions Menu	66
5.3	Log Cache	67
5.3.1	Rebuild Log Cache	68
5.3.2	Storage	68
6	Repository Profiles	69
6.1	Profiles	69
6.1.1	Add	69
6.1.2	Edit	71
6.2	Proxies	72
6.3	Tunnels	72
6.4	Passwords	72
7	Projects	74
7.1	Project Manager	74
7.2	Project Settings	75
7.2.1	Text File Encoding	75
7.2.2	Refresh/Scan	75
7.2.3	Working Copy	75
7.2.4	Default Settings	77
8	Subwindows	78
8.1	Text Editor	78
8.1.1	File menu	78
8.1.2	Edit menu	78
8.1.3	View menu	78
8.1.4	Go To menu	78
8.1.5	Window menu	78
8.2	File Compare	79
8.2.1	Comparison	79
8.2.2	File menu	79

8.2.3	Edit menu	79
8.2.4	View menu	80
8.2.5	Go To menu	80
8.2.6	Window menu	80
8.3	Conflict Solver	80
8.3.1	File menu	81
8.3.2	Edit menu	81
8.3.3	View menu	81
8.3.4	Go To menu	81
8.3.5	Window menu	81
8.4	Change Report (Pro Only)	82
8.4.1	File menu	82
8.4.2	Edit menu	82
8.4.3	View menu	82
8.4.4	Go To menu	83
8.4.5	Window menu	83
8.5	Log	83
8.5.1	Log menu	84
8.5.2	Edit menu	84
8.5.3	View menu	84
8.5.4	Modify menu	85
8.5.5	Query menu	85
8.5.6	Window menu	85
8.6	Revision Graph (Pro Only)	85
8.6.1	Graph menu	86
8.6.2	Edit menu	86
8.6.3	View menu	86
8.6.4	Modify menu	87
8.6.5	Query menu	87
8.6.6	Window menu	87
8.7	Annotate	87
8.7.1	Annotate menu	88
8.7.2	Edit menu	88
8.7.3	View menu	88
8.7.4	Revision menu (Pro Only)	88
8.7.5	Go To menu	88
8.7.6	Window menu	88
8.8	Merge Preview (Pro Only)	89
8.8.1	Merge menu	89
8.8.2	View menu	89
8.8.3	Window menu	89

9	Preferences	90
9.1	On Start-Up	90
9.2	Project	90
9.3	User Interface	90
9.3.1	Accelerators	91
9.3.2	Context Menus	91
9.4	Commit	91
9.5	Conflict Solver	92
9.6	Open	92
9.7	Refresh	92
9.8	Built-in Text Editors	93
9.8.1	View Defaults	94
9.9	File Comparators	94
9.10	External Tools	95
9.10.1	Directory Command	95
9.11	Transactions	95
9.12	Spell Checker	95
9.13	Shell Integration (Windows)	96
9.14	Shell Integration (Mac OS)	97
9.15	Check for Update	97
10	Shell Integration	98
10.1	Commands	98
10.2	Overlay Icons	98
10.3	Server Mode	99
10.4	Windows Shell Integration	99
10.5	Mac OS X Finder integration	100
10.6	Tray Icon	100
11	Installation and Files	102
11.1	Company-wide installation	103
12	System properties/VM options	104
12.1	General properties	104
12.2	SVN properties	104
12.3	User interface properties	106
12.4	Transaction-related properties	107
12.5	Other properties	108
12.6	Specifying VM options and system properties	108

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. Typically areas of application are software projects, documentation projects or website projects.

Acknowledgments

We want to thank all users, who have participated in the Early Access Program of SmartSVN and in this way helped to improve it by reporting bugs and making feature suggestions.

Special thanks goes to the SVNKit developers (<http://www.svnkit.com>) who provide the excellent Subversion base library SVNKit onto which SmartSVN has been built and of course to the whole SVN developer community at subversion.tigris.org for keeping Subversion one of the most powerful version control systems available today.

Used Notations

Throughout this help, features/operations which are only available in the Professional version will be marked by “(Pro Only)”.

Chapter 2

Project Window

The Project Window is the central place when working with SmartSVN. In the center of the window, the main **Directories** and **Files** view shows 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.

Always exactly one of the four views is “active” which is displayed by its 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.

The layout of the Project Window can be arranged with the mouse by dragging the splitters resp. the **Output** or **Transactions** view. By dragging their titles, they can be undocked from one position and docked to another position. Both views provide toolbar buttons to maximize and minimize resp. auto-hide them. The layout of the Project Window can be managed per project and it is stored and recovered for future SmartSVN sessions.

At the very bottom of the Project Window the status bar displays various kinds of information. The first and largest section 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 whether you are in file or in directory mode resp. in none of both if the **Directories** and **Files** view is not active. The third section displays information on the Refresh state (see 2.3.5) of the project and the fourth section is used for progress display during the execution of SVN operations. It may either show a percentual progress of the operation completion or the total amount of sent and received bytes during this operation.

2.2 Projects

SmartSVN internally manages your SVN working copies by “SmartSVN projects”, as basically described in Section 7.

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 by **Open** or closed by **Close**.

2.3 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 and *ignored* directories and files within other ignored directories are not displayed.

2.3.1 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. 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 gets active) and start typing the directory name. This will make a small popup come up, which displays the characters you have already entered. Wildcard symbols ‘*’ and ‘%’ can be used with the usual meaning.

2.3.2 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. Every primary state may be combined with additional states listed in Table 2.6. The rest of this section explains configuration options for the file table. They are always related only to the current project and are also stored with the current project.

File Attributes

Tip	Certain table columns require to access additional file system files when scanning the file system and therefore slow down scanning. The note within the View Table Columns dialog gives you information on 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 .txt extension.

From the attached drop-down menu you can reset the filter by **Show All** what simply clears the filter **Filter** field. You can also select to work with **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.

2.3.3 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 2.4.3) for details. By default, these file filters will also be applied on the directories. You can change this behaviour in the Preferences (see 9.3) by **Use View-menu file filters also for directories**.

2.3.4 Smart Open

When double-clicking a file in the file table, the file will be "opened" in one of various ways, depending on its file state:

- An *unchanged*, *unversioned* or *added* file is opened with the file editor, see the **Edit|Open** command (Section 2.4.2) for further details.
- A *conflicting* file is opened with the Conflict Solver (Section 8.3).
- All other files are opened by comparing them (Section 3.9.1).

2.3.5 Refresh

The contents of the directory tree and the file table are initialized when a project is opened by reading at least the contents of the root directory into memory. Whether the complete project shall also be read into memory at project startup or not can be configured in the project settings (Section 7.2).

The scanning and refreshing of the project's directories and files is in general 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 processed, the status bar shows a **Refreshing** text and symbol.

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 a priority in being scanned: Just select the (already scanned) directory in the **Directories** tree you would like to have scanned recursively as soon as possible. SmartSVN will then reorganize its breadth-first strategy accordingly.

When changes to known (i.e in memory) files or directories occur from within SmartSVN, they are refreshed automatically. In case of external changes, an explicit refresh via **View|Refresh** or by the corresponding toolbar button is required. You can configure the kind of refresh (“depth”) within the application preferences (Section 9.7).

2.4 Menus

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

2.4.1 Project

- **Create from Directory**, see Section 7.
- **Check Out**, see Section 3.1.
- **Create Module**, see Section 3.2.
- **Open**, see Section 7.
- **Close**, see Section 7.
- **Project Manager**, see Section 7.1.
- **Settings**, see Section 7.2.
- **Default Settings**, see Section 7.2.
- **Exit** exits SmartSVN.

2.4.2 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 other actions are not applicable.
- **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 9.10). If the file table has the focus the file(s) will be opened in an editor. The editor which shall be used to open a file can be configured in the **Externals Tools** section of the Preferences (see Section 9.10). If no editor is configured there, the internal File Editor (see 8.1) will be launched. For files, you can specify a limit on the file count beyond which you will be asked before the files are opened at once; for details refer to Section 9.6.

- **Select Committable Files** (Pro Only) selects all committable files in the file table. Because SmartSVN allows to automatically add *unversioned* or remove *missing* files for a commit, such files are also selected.
- **Select Directory** (Pro Only) selects the deepest common directory for all selected files in the file table.
- **Copy Name** (Pro Only) 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** (Pro Only) 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** (Pro Only) 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** (Pro Only) 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 Message** (Pro Only) 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** view. If there are multiple command outputs, the latest (topmost) command output will be kept. If there is only one command output, it will be removed. Currently processing commands can't be cleared.
- **Preferences** shows the application preferences (see Section 9.15).

2.4.3 View

- **Table Columns** lets you specify which file attributes will be displayed in the file table, see Table 2.3 and Table 2.4. Also, the order of the table columns can be defined here, alternatively to rearranging them directly in the file table. Select **Make this configuration the default** to have the selected configuration applied to every new project. Use **Reset** to reset the table column layout to the default.
- **Refresh**, see Section 2.3.5.
- **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 sometimes convenient to hide them, as they don't matter for most of the SVN commands.
- With **Unversioned Files** *unversioned* files (also within unversioned directories) are displayed.

Note	Unversioned Files option does in no way affect the unversioned files itself or their SVN state. Certain operations, which can work on unversioned files, will consider them anyway. Parent directories of unversioned files will continue to display <i>Direct/Indirect Local Changes</i> state. To actually ignore such files on the SVN-level you can use the Ignore command (Section 3.4.3).
-------------	--

- With **Ignored Files** ignored files within versioned directories will be displayed. Files from ignored directories are never displayed.
- With **Files Assigned to Change Set** (Pro Only) selected, files which are already assigned to a Change Set (see 3.12) will be displayed. Otherwise, those file will not be displayed, to give a better overview on which files are not yet assigned to 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** (Pro Only) selected, files will be displayed which are remotely changed (see Table 3.3). Typically, this option has no effect when **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.
- **Reset Layout** will reset the Project Window layout (docking states).

2.4.4 Modify

- **Update**, see Section 3.3.1.
- **Update More**, see Section 3.3.2.
- **Switch**, see Section 3.3.3.
- **Relocate**, see Section 3.3.4.
- **Merge**, see Section 3.6.1.
- **Merge from 2 Sources**, see Section 3.6.2.
- **Reintegrate Merge**, see Section 3.6.3.
- **Commit**, see Section 3.5.
- **Add**, see Section 3.4.1.
- **Remove**, see Section 3.4.2.
- **Ignore** (Pro Only), see Section 3.4.3.
- **Delete Physically**, see Section 3.4.4.
- **Create Directory** (Pro Only), see Section 3.4.5.

- **Rename**, see Section 3.4.6.
- **Move**, see Section 3.4.7.
- **Detect Moves** (Pro Only), see Section 3.4.8.
- **Copy**, see Section 3.4.9.
- **Copy From Repository**, see Section 3.4.10.
- **Copy To Repository**, see Section 3.4.11.
- **Copy Within Repository**, see Section 3.4.12.
- **Revert**, see Section 3.4.13.
- **Mark Resolved**, see Section 3.4.14.
- **Clean Up**, see Section 3.4.16.
- **Fix** (Pro Only), see Section 3.4.17.

2.4.5 Change Set (Pro Only)

- **Move to Change Set**, see Section 3.12.1.
- **Move Up**, see Section 3.12.2.
- **Move Down**, see Section 3.12.3.
- **Delete**, see Section 3.12.4.
- **Edit Properties**, see Section 3.12.5.

2.4.6 Tag/Branch (Pro Only)

- **Add Tag**, see Section 3.8.2.
- **Add Branch**, see Section 3.8.3.
- **Tag Browser**, see Section 3.8.4.
- **Configure Layout**, see Section 3.8.5.

2.4.7 Query

- **Show Changes**, see Section 3.9.1.
- **Compare with Revision**, see Section 3.9.2.
- **Compare 2 Files** (Pro Only), see Section 3.9.3.
- **Log**, see Section 3.9.4.
- **Revision Graph** (Pro Only), see Section 3.9.5.
- **Annotate**, see Section 3.9.6.
- **Create Patch**, see Section 3.9.7.
- **Create Patch between URLs**, see Section 3.9.8.
- **Refresh Remote State** (Pro Only), see Section 3.11.1.
- **Clear Remote State** (Pro Only), see Section 3.11.2.

2.4.8 Properties

- **Edit Properties**, see Section 3.7.1.
- **MIME-Type**, see Section 3.7.2.
- **EOL-Style**, see Section 3.7.3.
- **Keyword Substitution**, see Section 3.7.4.
- **Executable-Property**, see Section 3.7.5.
- **Externals**, see Section 3.7.6.
- **Ignore Patterns**, see Section 3.7.7.
- **Bugtraq-Properties** (Pro Only), see Section 3.7.8.
- **Merge Info**, see Section 3.7.9.

2.4.9 Locks

- **Refresh**, see Section 3.10.1.
- **Lock**, see Section 3.10.2.
- **Unlock**, see Section 3.10.3.
- **Show Info**, see Section 3.10.4.
- **Change 'Needs Lock'**, see Section 3.10.5.

2.4.10 Repository

- Use **Open in Repository Browser** (Pro Only) to open the selected directory/file in the Repository Browser (see 4).
- **Manage Profiles**, see Section 6.
- **Change Master Password**, see Section 6.4.
- **Repository Setup**, see Section 3.13.3.

2.4.11 Tools

- **Export Backup** (Pro Only), see Section 3.13.1.
- **Conflict Solver**, see Section 3.13.2.
- **Canonicalize URLs**, see Section 3.13.4.
- **Remove Empty Directories** (Pro Only), see Section 3.13.5.
- **Rebuild Log Cache**, see Section 3.13.6.

2.4.12 Transactions

- **Refresh**, see Section 5.2.
- **Mark as Read** (Pro Only), see Section 5.2.
- **Mark All as Read** (Pro Only), see Section 5.2.
- **Show Branches and Tags** (Pro Only), see Section 5.2.
- **Show Additional Watched URLs** (Pro Only), see Section 5.2.
- **Ungrouped Revisions**, see Section 5.2.
- **Grouped by Weeks**, see Section 5.2.
- **Grouped by Time**, see Section 5.2.
- **Grouped by Authors**, see Section 5.2.
- **Rollback** (Pro Only), see Section 8.5.4.
- **Change Commit Message** (Pro Only), see Section 8.5.4.
- **Configure Watched URLs** (Pro Only), see Section 5.2.
- **Settings**, see Section 5.2.

2.4.13 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** (Pro Only) shows the standalone Transactions Frame (see 5.1).

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.

2.4.14 Help

- **Help Topics** shows the online version of SmartSVN's help.
- **Register** lets you upgrade SmartSVN to the Professional version. You will need to purchase a license file, but it's definitely worth the money!
- **License Information** shows information on your SmartSVN license and the licensing conditions for SmartSVN.
- **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.
- **Downgrade Working Copy** can be used to downgrade the working copy format of the currently selected directory (including all subdirectories) to Subversion 1.4 working copy format. A working copy downgrade should only be necessary if you are forced to continue working with pre-1.5 SVN clients. The downgrade stops for externals, so you may consider to explicitly downgrade these externals, too.
- Use **Obfuscate Log Cache** to remove potentially confidential information from a Log Cache so it can be sent to syntevo GmbH 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 9.15).
- **About SmartSVN** shows information on the current SmartSVN version.


















Icon	State	Details
	Unchanged	Directory is under version control, not modified and equal to its revision in the repository resp. 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 to be ignored.
	Modified	Directory itself is modified in its properties (compared to its revision in the repository resp. to its pristine copy.)
	Added	Directory is scheduled for addition.
	Removed	Directory is scheduled for removal.
	Replaced	Directory has been scheduled for removal and 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.
	Conflict	An updating command lead to conflicting changes in directories' properties.
	Incomplete	A previous update was not fully performed. Do an update again.
	Root	Directory is either the project root or an external root.
	Nested Root	Directory is a nested working copy root, but no external. Refer to the Fix command (see 3.4.17) on how to handle such directories.
	Obstructed	A file exists locally, but the pristine copy (resp. repository) expects it to be a directory. Please backup the file, then remove it and update the directory from repository.
	Remote	Directory only exists in the repository. This state is only used for the remote state command (see Section 3.11).
	Unscanned	Directory has not been scanned yet (see Section 2.3.5).

Figure 2.1: Primary Directory States







Icon	State	Details
	Switched	Directory is switched (compared to its parent); see Section 3.3.3.
	Locked	Directory is locked <i>locally</i> because an operation has been interrupted before. A Cleanup (see 3.4.16) 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 3.11.
	Indirect Remote Changes	There are remote changes to one of its files or to one of the subdirectories of this directory, see Section 3.11.

Figure 2.2: Additional Directory States

SmartSVN Name	SVN info	Description
Name	(same)	The file's 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 3.10)
Last Rev.	Last Changed Rev.	Revision, where 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 3.7.3)
EOL	svn:eol-style	End-Of-Line Type of the file (see Section 3.7.3)
Keyw.	svn:keywords	Keyword substitution options of the file (see Section 3.7.4)
Needs Lock	svn:needs-lock	Whether the file should be locked before working (see Section 3.10.5)
Executable	svn:executable	Whether the file has the Executable-Property set (see Section 3.7.5)
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 (Pro Only)	Remote state of the file (see Section 3.11)
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 (Pro Only)	The tag/branch to which the file is currently switched (see 3.3.3). For details, refer to Section 3.8.1.
Change Set (Pro Only)	The Change Set (see 3.12) 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 resp. to its pristine copy.
	Unversioned	File is not under version control, but only exists locally.
	Ignored	File is not under version control (exists only locally) and is marked to be ignored.
	Modified	File is modified in its content or properties (compared to its revision in the repository resp. 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 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.
	Conflict	An updating command lead to conflicting changes either in content or properties.
	Incomplete	A previous update was not fully performed. You should do an update again.
	Remote	File does not exist locally, but only in the repository. This state is only used for the remote state (see Section 3.11).
	Obstructed	A directory exists locally, but the pristine copy (resp. repository) expects it to be a file. Please backup contents of the directory, then remove it and update the file from repository.
	Case-Changed	The case of the file name has changed on an operating system, which is case-insensitive regarding file names. Refer to the Fix command (see 3.4.17) on how to handle such files.
	Merged	The file has been merged. Refer to the Merge command (see 3.6.1) for details.

Figure 2.5: Primary File States





Icon	State	Details
	Switched	File is switched (compared to its parent directory); see Section 3.3.3.
	Locked (Self)	The file is locked in the repository by yourself (resp. for the current working copy), see Section 3.10.
	Locked (Other)	The file is locked in the repository by some other user, see Section 3.10.
	Lock Necessary	The file needs to be locked before starting to work, see Section 3.10.5.

Figure 2.6: Additional File States

Chapter 3

Commands

SmartSVN provides most of the SVN command line commands in a standalone version, but also combines them to powerful higher-level commands. Common enhancements, which are present for various 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”

If you are going to frequently check out from a repository you may perform a **Detailed Checkout**. First you need to select the repository from which you want to check out a project. If you can't find the **Repository Profile**, click the **Manage** button to add it, see [Section 6](#) for details.

The **Quick Checkout** is similar to the command line version of checkout: Simply enter the **URL** of the project you want to check out and specify the **Local Directory** to check out to. The subsequently described steps are skipped, when using the **Quick Checkout**.

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 more details refer to [Section 4](#).

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

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

Tip	When working with <i>trunk</i> , <i>tags</i> and <i>branches</i> it's not recommended to check out the whole project, because due to the rising number of tags the working copy (not the repository!) would be growing rather fast, containing a lot of useless files on your local disk. Instead you should check out only <i>trunk</i> or a certain tag or branch and if necessary switch (see 3.3.3) to another location.
------------	---

Page “Target Directory”

At 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 by Update More (see [3.3.2](#)). Non-recursive checkouts (also called “sparse checkouts”) can be useful, if you wish to skip certain *modules* of a project.

Click **Next** to proceed.

Page “Project”

At this page you can select whether to check out a working copy, i.e. create the necessary `.svn/` structure or to simply **Export** the files from the repository. With **Check out a working copy and manage as project**, SmartSVN will add the working copy to its list of projects, using the specified **Project Name**. In case of **Export only**, you have to specify the desired line endings marker with **Use EOL**. When selecting **Overwrite locally existing files**, locally existing files will be overwritten if necessary, otherwise the export will be cancelled.

Click **Next** to proceed.

Page “Confirmation”

Use this page to review your choices. Click **Back** to change them or **Finish** to start the checkout.

3.2 Create Module

Use **Project|Create Module** to add a new locally available project to the repository and to create the corresponding SmartSVN project.

Page “Directory to Import”

Select the **Local Directory**, for which you want to create a new project and a new module in the repository.

Page “Repository”

Choose the **Repository** you want the new module to be created in. If the **Repository Profile** does not exist yet, click the **Manage** button to add it.

Page “Location”

After switching to this page, it takes a few moments until the repository is scanned. You can now scan into the repository by expanding the directory nodes, for more details refer to Section 4. Use **Create Directory** to create a new directory for your project in the repository.

Note	You can create recursive directories at the same time, by specifying the directories separated by /. This helps to keep the Log nicer as there will only show up one transaction for creating all the directories.
-------------	--

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

Page “Project Name”

At this page you can assign a **Project Name** for the project, which will automatically be created. If you just want to import the project without further working with it, deselect the option **Add to list of managed projects**.

Page “Confirmation”

Use this page to review your choices. Click **Back** to change them or **Finish** to start the Module creation.

Configuring the project and doing the final import

The result of the Create Module 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 by the corresponding 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 Updating

Updating from the repository can happen either by a simple update of the working copy or by switching the working copy to another location/revision. Following commands are available from the **Modify** menu.

3.3.1 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.

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 before.

3.3.2 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 not yet checked out subdirectories resp files. 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 shall be updated resp. checked out recursively or not.

3.3.3 Switch

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

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

You can either switch to the selected location **At** or at a specific . Select **Recurse into subdirectories** to perform the switch command not only for the currently selected directory, but also for all subdirectories. Regarding **Allow unversioned obstructions**, refer to the Update command (see 3.3.1).

3.3.4 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**.

3.4 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. Following local commands are available from the **Modify** menu.

3.4.1 Add

Use **Modify|Add** to schedule files or directories for being added 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 3.7.2), which can basically be *text* or *binary*. Further property defaults can be specified in the project settings (see 7.2).

Tip Automatic detection can be overridden by the **Binary Files** project settings (see Section 7.2.3).

3.4.2 Remove

Use **Modify|Remove** to schedule the selected files/directory for being removed 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, which have local modifications or directories which contain modified or unversioned files. Select **Force Removal** if you wish to perform the removal of such items anyway.

3.4.3 Ignore (Pro Only)

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

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, it's necessary that its parent directory is either ignored too or is versioned, so 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. Contrary to *local ignore patterns* you can configure *global ignored patterns* in the project settings (see 7.2).

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 3.7.7 for details.

3.4.4 Delete Physically

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

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

3.4.5 Create Directory (Pro Only)

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

Enter the **Path** of the subdirectory, which shall be created. The path may consist of multiple directory names, separated by “\” resp. “/” to create multiple directories at once. Select **Schedule for addition** to schedule the created directory/directories for being added to SVN control, see Section 3.4.1.

3.4.6 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.

3.4.7 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 commands, which works on exactly two selected files, where one of the files is missing or removed and the other one is unversioned, added or replaced. SmartSVN interprets this as a “post-move”, removes the missing (if necessary), adds the unversioned file (if necessary) and connects the history of the added file to that of the removed file.

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

3.4.8 Detect Moves (Pro Only)

Use **Modify|Detect Moves** to convert already performed “manual” moves (including renamings) of files to “SVN” moves. Typically, you will not perform moves within SmartSVN itself, but with system commands, by IDEs, etc. One such external move results in a missing and a new unversioned file. Both files could then be added resp. removed and committed, what will result in a correct repository content, but will not preserve the relation 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.

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 would be equal to **Target Path**.

If you agree that such a **Target-Source** pair actually represent 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 purely based on the comparison of the file contents and denotes the calculated likelihood for the file pair representing an actually happened move. 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. that file with yields the highest **Similarity**, and **Alternatives** shows the number of possible alternative sources. Use **Alternatives** to select an alternative source to be used instead of the original suggestion. Finally, if you consider a suggestion and all available **Alternatives** not correct, you may deselect the suggestion so no “post-move” will be performed for that specific **target**.

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

3.4.9 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** and the new **File Name** under which the copy of the file/directory shall be created.

There is also a special mode of this commands, which works on exactly two selected files, where one of the files is versioned, but not added or replaced and the other one is unversioned, added or replaced. SmartSVN interprets this as a “post-copy”, adds the unversioned file (if necessary) and connects the history of the added file to that of the other file.

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

3.4.10 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 is useful to resurrect dead files or 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 and its **Source Revision** which shall be copied. Specify the local directory **Into Local** into which the file/directory shall be copied. **With Name** will be the actual name (last component of the path).

3.4.11 Copy To Repository

With **Modify|Copy To Repository** you can copy the selected local file/directory to the repository. This operation is useful for creating tags, although SmartSVN provides more convenient functions for this task (see Section 3.8).

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 actual name (last component of the path). Because the copy is directly performed into the repository, you have to specify a corresponding **Commit Message**.

Use **Externals Revisions** (Pro Only) to specify how to handle externals revisions (see 3.7.6). This option is only relevant for externals which have their revisions set to **HEAD**. By default, **Leave as is** will not modify any externals 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 externals which are in general not equal to that at the time of creating the copy.

3.4.12 Copy Within Repository

With **Modify|Copy Within Repository** you can perform pure repository copies. This is for instance a convenient and fast way to create repository tags/branches.

Select the **Repository** within which the copy shall occur. **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 case of copy **To**, the source will be copied into **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 actually no local operation, as it requires no working copy. For convenience we have put it into the chapter “Local Modifications” anyway.

3.4.13 Revert

Use **Modify|Revert** to revert the local changes of the selected files/directories. In case of directories you have the option 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 return 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 3.3.1) 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!).

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

3.4.14 Mark Resolved

Use **Modify|Mark Resolved** to mark conflicting files/directories 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 old** to replace the contents of resolved files by the contents of their corresponding pristine copies as they were before the update/merge. Use **Take new** to replace the contents of resolved files by the contents of their corresponding pristine copies as they are now after the update/merge. Use **Take working copy** to replace the contents of resolved files by their contents as they were before the update/merge.

3.4.15 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 express that the content of a directory/files is not related to its previous revision. The Log (see 3.9.4) 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 rename 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 of that one 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 3.4.7), yielding a *removed* `PersonImpl.java` and a *replaced* `Person.java`, which has its history (**Copy From**) set to `PersonImpl.java`.

3.4.16 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 gets “internally” locked (in contrast to file locks, see Section 3.10). A working copy can get 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/written. In this case, please check whether other running processes might lock the file and whether file permissions have been set adequately.
-------------	--

3.4.17 Fix (Pro Only)

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

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 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).

Anyway, 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 *case-changed* file state, see Table 2.5. **Fix** will now change back the file name case 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 are typically resulting from moving a directory from one location to another one, without using appropriate SVN commands, like Move (see 3.4.7). This leaves a *missing* directory at its original location and introduces *nested root* at its current location.

Fix offers 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 SVN admin area (.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.

3.5 Commit

Use **Modify|Commit** to write back (commit) the changes of the selected files/directory to the repository.

The **Commit** wizard guides you through the commit, starting with the "Configuration". Based on the "Configuration" the working copy will be scanned for changes, this is especially important when performing the **Commit** on a directory. Subsequent pages allow further "customization" of the commit. Their presence depends on the changed files and directories found during the scanning phase.

Before the commit wizard opens, it checks whether you might have missed to select some files resp. directories and in this case shows a warning. For details, refer to the Preferences (see [9.4](#)).

Page “Configuration”

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** (Pro Only) to also scan for changes in external working copies (see [3.7.6](#)). Select **Skip Change Set entries** (Pro Only) to ignore found changed files resp. directories which have already been assigned to a Change Set (see [3.12](#)).

Select **Detect moved and renamed files** (Pro Only) 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 [3.4.8](#).

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** (Pro Only) to also report unversioned (most likely new) files and directories. Select **Remove missing files and directories** (Pro Only) to also report missing (most likely obsolete) files and directories.

Note	When committing a selection of files, only those options are available, which are actually applicable to the files. So, for instance, when no unversioned files have been selected, Add unversioned files and directories will not be available.
-------------	---

Select **Remove removed parent directories** (Pro Only) 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** (Pro Only), 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 3.13.5 .
------------	---

When clicking **Next** the file system of your project will be scanned, especially in case of directory commits, this may take some time.

Page “Detect Moves” (Pro Only)

This page is only displayed if the option **Detect moved and renamed files** on the **Configuration** page has been selected and at least one moved or renamed file pair was detected. By **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 navigation for this view is similar to the Change Report (see [8.4](#)).

Page “Externals” (Pro Only)

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 3.7.6.

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 checkbox in **Path** column (either with the mouse or with *<Space>*).

Working copies pointing to the same repository (the **URL** is helpful to see this) 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 is not sustainable. 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 its working copy and only have received the successfully committed revision. This might result in (temporarily) inconsistencies of his/her overall project.

You can choose whether to commit the selected working copies with **One commit message** or with **Individual commit messages**. This choice may not be available even when having multiple working copies selected. This happens if the Bugtraq-Properties (see 3.7.8) for at least two of the working copy roots are set and these properties are not identical.

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. To skip a file/directory from commit, deselect the corresponding checkbox (either with the mouse or by pressing *<Space>*).

You may review your changes, by expanding the dialog with the **Differences** (Pro Only) button. This will show the differences for the currently selected file in the lower part of the **Commit** dialog. The navigation for this view is similar to the Change Report (see 8.4).

For the **Commit Message** you can enter arbitrary text or select an older message from the message popup right to the text field. The popup menu will show up recently entered commit messages, allow to clear this message history by **Clear Message History** or use **Get from Log** (Pro Only) to fetch an older commit message from the log. By *<Ctrl>+<Space>* you can trigger a file name completion, based on all of those files which have been selected for the commit.

Depending on whether resp. how Bugtraq-Properties (see 3.7.8) 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 9.12) has been configured, SmartSVN will check the entered **Commit Message** for basic spelling errors. 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-Properties.

Tip	Commit messages will be displayed in various kinds of logs. Hence, a meaningful commit message is very helpful for you and your team to track your changes.
------------	---

By default, SmartSVN will warn you in case of an empty commit message. You can switch this warning off in the Preferences (see 9.4).

If **Descend into externals** has been selected and multiple working copies on the **Externals** page have been chosen to commit, there will also be a topmost **Working Copy** selector. All other items on this page are always related to the selected working copy. In particular it will be necessary resp. it is recommended to enter a **Commit Message** for each working copy.

Page “Locks”

This page will only be displayed, if the selected files/directories for the commit, which have been scanned, contain locked files.

Select **Keep locks for committed files** to keep the files locked even after having them committed. Select **Unlock committed files** (Pro Only) to unlock them after the commit. In this case you can also select further unchanged but locked files which had been detected during the scan and which shall 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 7.2.3).
------------	---

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

3.6 Merging

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

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

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

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

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

Warning! As merging can become a rather sophisticated task, there are certain recommendations. The most important ones are:

- Do only recursive merges and try to always merge on the same “merge root”, preferably *trunk* itself or the root of a branch.
- Avoid merging into a working copy which contains mixed revisions. Therefore do an Update (see 3.3.1), preferably to **HEAD**, before.
- Avoid merging into non-recursively (resp. non-completely) checked out working copies. Therefore do an Update More (see 3.3.2) on your merge root, selecting all files and directories and the **Recurse into subdirectories** option.

3.6.1 Merge

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

Select **Trunk** (Pro Only) to merge from the main trunk. Select **Branch** or **Tag** (Pro Only) and enter the branch or tag name to merge changes from a branch or tag. Select **Other URL** to merge from an arbitrary URL, specifying the corresponding repository and **Path**.

Use **All Revisions** to merge all those 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*.

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 to repeatedly merge them. 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 resp. ranges can be specified by separating them by a colon (,). Instead of entering the revisions manually, you can choose them from the revision browser (see 3.14.2). The revision browser will also display those revisions which have not been merged by a green arrow.

Example

You will typically use this option when working with a *release branch* to get 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. This means, 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 behaviour leads to the expected results and it is also required for the merge tracking to work. You can switch this behaviour off by selecting **Ignore ancestry**, however this option is not recommended unless you have a good reason to use it.

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, but only the Mergeinfo (see 3.7.9), will be adjusted correspondingly, so 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** (Pro Only) the changes which the merge will bring, for details refer to the Merge Preview (see 8.8).

Tip	You can choose to keep the auxiliary merge files even for non-conflicting files in the Section 7.2.3.
------------	---

3.6.2 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** selector. For details regarding the **Advanced** options, refer to Section 3.6.1.

Note	Most merging use cases are covered by Merge (see 3.6.1) and Reintegrate Merge (see 3.6.3) and if possible these commands should be used.
-------------	--

3.6.3 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 of a *feature branch* has been finished and the “feature” shall 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, etc. 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 3.6.1.

3.7 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. Following commands are related to properties and are available from the **Properties** menu.

3.7.1 Edit Properties

Use **Properties|Edit Properties** to display and edit properties of the selected file/directory.

The table displays all properties of a file/directory with their **Name**, **Current Value** and **Base Value** (the value from the pristine copy). Similar to the File Compare, changes in property values between current and base value are highlighted by different colors.

To enter a new property, select the last (empty) line in the table, enter the property’s name and its value, then use **Add**. To create a copy from another property, select the original property, change the property’s name (and maybe its value), then use **Add**. To change a property, select it in the table, change the name or the value and use **Apply**. To delete a property, select it and use **Delete**.

Note	Internal SVN properties starting with svn can't be edited directly. Instead, SmartSVN offers special commands in the Properties menu to modify them.
-------------	--

3.7.2 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 case of *text* types the line endings, etc. can be replaced. By default, when adding files (see Section 3.4.1), 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 7.2.3) you can define file name patterns which should always be treated as *binary*.

3.7.3 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 file and results in a corresponding conversion of its line endings:

- **Platform Dependent** converts to the platform's native line separators.
- **Unix, Mac, Windows** converts to the corresponding line separators, regardless of the current platform.
- **As is** performs no conversion.

In the project settings (see 7.2.3), the default EOL-style which will be applied to every added file can be specified. 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 to change the EOL-style (other behaviors can be configured in the project settings). To skip this check, use **Force**.

3.7.4 Keyword Substitution

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

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

3.7.5 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.

3.7.6 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. 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**. Enter `bazz` into the **Local Path** input field, `http://server/svn/foo` into the **URL** input field, 4711 to the **Revision** input field and click **Add**. 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 actual version you are working with. When you decide to use a more recent revision of the external, you can evaluate it before and if you are satisfied, increase the Revision number of the external definition.
------------	---

3.7.7 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 to specify *global ignore patterns* within the project settings (see 7.2.3).

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

3.7.8 Bugtraq-Properties (Pro Only)

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://tortoisesvn.tigris.org/svn/tortoisesvn/trunk/doc/issuetrackers.txt>, username is **guest** with empty password. SmartSVN implements this specification with following mapping from UI elements to core `bugtraq:-properties` as shown in Table 3.1.

bugtraq-Property	UI Element
<code>bugtraq:url</code>	URL
<code>bugtraq:warnifnoissue</code>	Remind me to enter a Bug-ID
<code>bugtraq:label</code>	Message Label
<code>bugtraq:message</code>	Message Pattern
<code>bugtraq:number</code>	is true exactly if Bug-ID is set to Numeric
<code>bugtraq:append</code>	is true exactly if Append message to set to Top
<code>bugtraq:logregex</code>	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

Your commit messages look like: Ticket: 5 Some message or ticket #5: Some message and you want the 5 show up 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 show up as a link, use the same **Bug-ID expression** and also set **Message-Part Expr.** to this value.

3.7.9 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 “block” certain revisions manually from being merged, you should use **Modify|Merge** with the **Record only** option set.

3.8 Tags and Branches (Pro Only)

SmartSVN simplifies the handling of “Tags” and “Branches”. Both “Tags” and “Branches” are no native SVN concepts, but can easily be handled by the help of Copy To Repository (see 3.4.11) and Copy Within Repository (see 3.4.11). 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.

3.8.1 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. It affects the presentation of and the working with URLs for various commands. 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 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 resp. tag root directories. All paths are relative to the **Project Root** and when using values **trunk**, **branches/*** and **tags/*** here, you will be compatible with the suggested 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 resp. tag patterns. In this case, when entering e.g. 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** following patterns should be used: `branches/*`, `shelves/*/*`.

Now, when e.g. creating a branch “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.

When e.g. 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.

3.8.2 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. Such a tag will contain local changes, if present. It 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 3.3.1) to that revision before and make sure that there are no switched directories.
------------	---

By default, SmartSVN will fail if the specified tag already exists. Select **Overwrite existing tag, if necessary** to create the tag anyway, replacing the already existing tag.

Use **Externals Revisions** (Pro Only) to specify how to handle externals revisions (see 3.7.6). For details refer to Section 3.4.11.

Note	This command is similar to Modify Copy Local to Repository (see Section 3.4.11), but simplifies the special task of “Tagging”.
-------------	---

3.8.3 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 3.8.2 for details.

3.8.4 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/branches location as specified in the Tag/Branch Layout (see 3.8.5). The subsequent table will contain tags and branches found herein. A tag resp. branch has a **Name**, a **Revision** at which it had been created and possibly 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.

From the **View**-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.

Use **Removed Tags/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.

Tip	You can invoke the Tag Browser also from tag or branch name input fields by clicking the ellipsis button to the right (...).
------------	--

3.8.5 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 3.7.6). For details refer to Section 3.8.1.

3.9 Queries

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

3.9.1 Show Changes

Use **Query|Show Changes** to compare the selected files resp. all files within the selected directory against their pristine copies. **Show Changes** will correspondingly open one or more File Compare (see 8.2) frames or the Change Report (see 8.4) frame. For details, regarding the warning limit on the number of files to compare at once, refer to Section 9.6. No connection to the repository is required.

3.9.2 Compare with Revision

Use **Query|Compare with Revision** to compare a single, local file with another revision of the file. Select either to **Compare** the **Working Copy** or the **Pristine Copy**. Select to compare **With** the **Trunk** or a specific **Branch** or **Tag** (Pro Only) or an arbitrary

Other URL. Select whether to retrieve the repository file **At** the repository **HEAD** or at a another **Revision**. The result will be a File Compare (see 8.2) frame.

3.9.3 Compare 2 Files (Pro Only)

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.

3.9.4 Log

Use **Query|Log** to display the change history of the selected file/directory. On the **Configuration** page you can specify, how far back in history the changes should be displayed.

Select **Stop logging on copied locations**, to make SmartSVN not trace further changes after it has encountered a revision where the file/directory has been copied from another location.

Select **Include merged revisions** to also fetch the originating revisions for revisions which 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 result in a large or even huge number of reported revisions.

On the **Advanced** page, you can configure the usage of the Log Cache (see 5.3). By default, the Log Cache is **Enabled with updating**, which will speed up logging. You can also choose **Enabled without updating** to skip updating the cache from the repository, before it is queried. With this option you can perform logs without requiring any connection to the repository. However new revisions from the repository won't be displayed. With **Disabled** the log command will be performed directly against the repository. This can be helpful if your Log Cache is obsolete due to changes in the repository of already cached log data, see Section 5.3 for details.

Note	When using Include merged revisions with the Log Cache being Enabled , it will still be necessary to perform the Log directly against the repository if the Log Cache has not been built with <i>mergeinfo</i> . The Log may also be performed directly against the repository if the corresponding Log Cache is currently updating a large number of revisions from the repository. The reason is that instead of waiting for the Cache update to be finished it will in general be faster to perform the Log directly.
-------------	---

When **Log HEAD instead of working revision** is selected, the Log will be performed against the selected directory's/file's URL at HEAD. This will report even revisions for the URL which are newer than the corresponding working copy revision. The disadvantage of this option is that the Log might fail, because the URL does no longer exist within the repository at HEAD.

After you have configured the command, click **OK** to proceed. Depending on the configuration the upcoming **Log** frame will show the resulting log as a directory/file tree or as a list of single file revisions. For details refer to Section 8.5.

3.9.5 Revision Graph (Pro Only)

Use **Query|Revision Graph** to display the complete “family tree” for the selected file or directory. The Revision Graph shows all entries (files/directories) within all revisions which are related to the selected file/directory, either by subsequent changes or by *copies*, in both directions future and past. Hence, the Revision Graph of an entry also contains the complete Log (see 3.9.4) of that entry upto its origin.

On the **Configuration** page, you can configure the **Log Scope** of the Revision Graph: The Revision Graph is based on the complete log of a subset of the repository (not only the file/directory itself), what is necessary to trace *copies*. Unfortunately, logging the complete repository can require significant computational effort, even if the Log Cache (see 5.3) is used. On the other hand, entries are typically not copied across the whole repository, but only across a certain part of it. This is e.g. the case, when creating a branch, or moving one file from one directory to another. Using this knowledge, you can limit the computational effort by only logging the **Project Root** of your current project or even a specific **Path** instead of the whole **Repository Root**. If you are using one of the former two options and the file had been copied from/to a revision out of the scope anyway, the Revision Graph will simply not trace this link and hence skip the corresponding sub-tree.

When creating a Revision Graph for a directory, you can also choose to **Report Children**. This will not only show revisions, where the directory itself has been modified (properties), but also all revisions, where one of its (in)direct children has been modified. Revisions, where children have been modified are considered as simple “modifications” of the directory, independent of whether children have been modified/added/removed or copied.

Warning! Be careful, when using **Report Children**, because – depending on the selected directory and the size of your repository – this can result in really huge revision graphs and a correspondingly large memory consumption.

On the **Advanced** page, you can configure the usage of the Log Cache (see 5.3), see the Log command (Section 3.9.4) for details.

After performing this command, the **Revision Graph** window of the selected file/directory will come up. For details, refer to Section 8.6.

3.9.6 Annotate

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

Similar to the Log command (see Section 3.9.4), you can configure the period of time for which the annotated view shall be calculated.

On the **Advanced** page, select **Track content of all revisions** (Pro Only) to have the file contents of all revisions present for the subsequent **Annotate** window. Otherwise you will only be able to see the content of the latest revision for the selected file.

Use **Treat even binary revisions as text** to continue the Annotate even when it encounters one or more binary revisions of the file. This option can be necessary if the MIME-Type (see 3.7.2) of a file had been corrected e.g. from *binary* to *text* in some earlier revision, although the file had *text* content since ever. In case the file actually had binary content in some earlier revision, parts of the annotate output might be trash.

After performing this command, the **Annotate** window for the selected file will come up. For details refer to Section 8.7.

3.9.7 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.

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

On the **Advanced** page, select **Ignore change in EOL-Style** to not output line changes for which only the line ending differs. This can be useful after having the line endings of a local file changed temporarily, but only “relevant” changes should be part of the patch.

With **For Whitespaces** you can configure to not output certain changes which are only affecting whitespaces. Use **No special handing** to do not ignore any changes regarding whitespaces. Use **Ignore changes in the amount** to ignore those 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 output only lines where anything else but whitespaces has changed.

3.9.8 Create Patch between URLs

Use **Query|Create Patch between URLs** to create a “Unidiff” patch between two arbitrary URLs. See Section 3.9.7 for more details on patches.

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. This means, that 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 decide to switch this behaviour off by selecting **Ignore ancestry** on the **Advanced** page. For details regarding the other **Advanced** options, refer to Section 3.9.7.

3.10 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 2.6. For the list of possible lock states, refer to Table 3.2.

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 meanwhile it has been stolen by someone other in the repository.
Broken	The file was locked for the local working copy, but in the meanwhile it has been unlocked (by someone other) in the repository.
(Username)	The file is currently locked by the named user.

Figure 3.2: Lock States

The “Self” state can be filled by SmartSVN when scanning the local working copy. Please note, that this state can change, when scanning the repository (see Section 3.10.1), as the lock might actually be “Stolen” or “Broken”.

3.10.1 Scan Repository

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

You can combine scanning the repository for locks with refreshing the Remote State (see 3.11) in the Preferences (see 9.7). You can also schedule a recurrent refresh of the repository lock information in the Project Settings (see 7.2.3).

3.10.2 Lock

With **Locks|Lock** you can lock the selected files in the repository. You can enter a **Comment**, 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). In this way it can happen that you “steal” the lock from another user, what can lead to confusion, when the other user continues working on the locked file. Hence you should use this option only if necessary (e.g. if someone is on holiday and has forgotten to unlock important files).

3.10.3 Unlock

With **Locks|Unlock** you can unlock the selected files resp. 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. In this way it can happen that you “break” the lock from another users, what can lead to confusion, when the other user continues working on the locked file.

3.10.4 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 is set (see Section 3.10.5). **Comment** is the lock comment, as entered by the user at the time of locking.

3.10.5 Change ‘Needs Lock’

With **Locks|Change ‘Needs Lock’** files can be marked/unmarked to require locking. This is useful to indicate 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.

3.11 Remote State (Pro Only)

The remote state signals, what would happen in case of an update to HEAD (see Section 3.3.1). 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
Latest	The local entry is equal to the latest revision of this entry in the repository. An update on this entry will bring no changes.
Will be modified	For the local entry there exists a newer revision in the repository. An update will bring the corresponding changes for this entry.
Will be removed	The local entry has been removed in the repository. An update will remove the entry locally.
Will be 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 to the directory tree resp. the file table, which do not exist locally. To such directories and files the special local state “Remote” is assigned, see Table 2.5 and Table 2.1.

3.11.1 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, to 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 3.10.1) in the Preferences (see 9.7) to have the Remote State automatically be 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.

3.11.2 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.

3.12 Change Sets (Pro Only)

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 2.3) 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 (resp. 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 3.12.1). 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 3.5).

Another convenient way to assign files to Change Sets is the local Change Report (see 3.9.1). When the project directory structure is selected (as opposed to a Change Set), deactivating **View|Files Assigned to Change Set** (Section 2.3) 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.
-------------	--

3.12.1 Move to Change Set

Use **Change Set|Move to Change Set** to change the assigned Change Set (see 3.12) 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* resp. 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**.

3.12.2 Move Up

Use **Change Set|Move Up** to move the selected Change Set (see 3.12) one position up (when having multiple Change Sets).

3.12.3 Move Down

Use **Change Set|Move Down** to move the selected Change Set (see 3.12) one position down (when having multiple Change Sets).

3.12.4 Delete

Use **Change Set|Delete** to delete the selected Change Set (see 3.12). This will only affect the Change Set assignment, not the files nor their SVN state.

3.12.5 Edit Properties

Use **Change Set|Edit Properties** to change the **Message** and other properties of the selected Change Set (see 3.12). For details, refer to Section 3.12.1.

3.13 Tools

The **Tools** menu offers several tools/utilities which can be useful when working with SVN projects.

3.13.1 Export Backup (Pro Only)

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

Root Directory displays the common root of all files to be exported and the exported file's paths will be relative to this **Root 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**.

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

Select **Include Ignored Files** resp. **Include Ignored Directories**, if you want to include these ignored items (and their contents) as well.

3.13.2 Conflict Solver

Use **Tools|Conflict Solver** to start a kind of *Three-Way-Merge*, which can be invoked on *conflicting* files (see Table 2.5). For details, refer to Section 8.3.

3.13.3 Repository Setup

Use **Tools|Repository Setup** 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 checks whether the chosen files are correct by executing svnadmin --version resp. svnserve --version . Later on SmartSVN needs to be able to execute svnadmin create [repository] resp. invoke svnserve -d -r [repository-root] .
-------------	---

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

On the **Username** page, enter a **Username** and **Password** 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 other users resp. 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 6.1.1 for details. Select **Proceed with importing files into the repository** to continue with the Create Module wizard (see 3.2).

3.13.4 Canonicalize URLs

Use **Tools|Canonicalize URLs** to rewrite URLs of `.svn`-files to canonical form, this means omitting default port numbers. Having an URL in canonical form is convenient, because you need not to enter the port number when working with the URL.

Select **Include Externals** to also canonicalize externals. Canonicalizing externals can require to rewrite the `svn:externals` property (Section 3.7.6). In this case the affected directories will be in *modified* state after the canonicalization and you have to commit them by yourself to finish the canonicalization.

3.13.5 Remove Empty Directories (Pro Only)

Use **Tools|Remove Empty Directories** to schedule 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.

3.13.6 Rebuild Log Cache

Refer to Section 5.3.1.

3.14 Common Features

SmartSVN includes a set of common features resp. UI elements, which are shared by various commands.

3.14.1 Recursive/Depth options

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

Alternatively, you can switch to *advanced* recursion options in the Preferences (see 9.3). In this mode SmartSVN offers the Subversion *depth* levels:

- **Empty** only operates on the directory itself.

- **Files** operates on the directory and its directly contained files.
- **Immediates** operates on the directory, its directly contained files and subdirectories, but not on files or directories within these subdirectories.
- **Infinity** operates on the directory and contained files and subdirectories recursively.
- **As Is** is only available for certain commands, like Update (see 3.3.1), and works on the directory, files and subdirectories as *they are*. If available, **As Is** is the recommended option to be used because it does not alter the *depth* of the directory structure.

Hence, having **Recurse into subdirectories** selected is equivalent to depth **Infinity** while having **Recurse into subdirectories** deselected is equivalent to depth **empty**.

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 by selecting the ellipsis (...) button after the input field.

A dialog displaying all revisions for the selected 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 3.8 for further details. The **Revision** column shows the revision number resp. the corresponding tag. The other columns display the revision's **Time**, **Commit Message** and **Author**, resp.

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.

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 selecting the ellipsis (...) button after the input field.

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 (Pro Only)

Input fields, for which you can enter a tag, like when using Switch (Section 3.3.3), support a *browse* function, which can be accessed by selecting the ellipsis (...) button after the input field.

The Tag Browser (Section 3.8.4) 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`.

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 **Repository|Open in Repository Browser** or by **Window|New Repository Browser**. If a tray icon (see 10.6) 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 Create Module (see 3.2) provide inbound Repository Browsers.

The Repository Browser displays the repository content with a **Directory** tree and a **File** table, similar to the Project Window (see 2). The repository file system is only scanned on demand. This happens when currently unscanned directories are expanded. The Tag-Branch-Layouts (see 3.8.1) 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. Use **Manage** to create a new Repository Profile (see 6) if necessary.
- Use **Show Revision** to change the currently displayed revision.
- Use **Check Out** (Pro Only) to check out the currently selected directory. This will bring up a simplified **Check Out** wizard, for details refer to Section 3.1.
- Use **Manage Profiles** to create a new Repository Profile (see 6) if necessary.
- Use **Close** to close the frame.

4.2 Edit menu

- Use **Stop** to cancel the currently processing operation. This action might not be applicable for certain operations.








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 3.8.1).
	Trunk/Branch	A <i>trunk</i> or <i>branch</i> , according to some Tag-Branch-Layout (see 3.8.1).
	Tag	A <i>tag</i> according to some Tag-Branch-Layout (see 3.8.1).
	Intermediate	An intermediate directory according to some Tag-Branch-Layout (see 3.8.1). 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

- Use **Open** to open resp. view 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 2.4.2) command in the Project Window (see 2).
- Use **Configure Layout** (Pro Only) to configure the Tag-Branch-Layout (see 3.8.1) for the currently selected directory.
- Use **Dismiss Layout** (Pro Only) 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.

4.3 View

- 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

- **Create Directory**, see Section 4.4.1.
- **Remove**, see Section 4.4.2.

- **Copy** (Pro Only), see Section 4.4.3.
- **Move** (Pro Only), see Section 4.4.3.

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

4.4.2 Remove

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

4.4.3 Copy/Move (Pro Only)

Use **Modify|Copy** or **Modify|Move** to copy resp. 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 don’t have manually modified it. Select **Reset Revision to HEAD** to reset the Repository Browser’s revision to HEAD after having performed the copy. This option is only available if the current revision 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 resp. move files and directories. This will open the same dialog with the corresponding paths pre-filled.
------------	--

4.5 Query menu (Pro Only)

- Use **Log** to display the log for the currently selected directory or file. For details refer to Section 3.9.4.
- Use **Revision Graph** to display the revision graph for the currently selected directory or file. For details refer to Section 3.9.5.

- Use **Annotate** to display an annotated view of the currently selected file's content. For details refer to Section [3.9.6](#).
- 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).
- Use **Show Properties** to display the properties of the currently selected file or directory.

4.6 Window menu

Refer to Section [2.4.13](#) 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, but also to allow similar powerful queries as the Log command (see 3.9.4) 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 (Pro Only)

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 10.6) 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. 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 3.8.1).
- **Revision Number:** Displays the revision number.
- **Time:** Displays date and time of the revision. The used format can be changed in the Preferences (see 9.3).
- **Trunk/Branch/Tag:** displays the corresponding *trunk*, *branch* or *tag* to which the revision belongs, refer to Section 3.8.1 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 revision's file count.

5.1.1 Grouping of revisions

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

- Ungrouped
- Weeks
- Time
- Authors

5.1.2 Watched URLs

Use **Edit|Configure Watched URLs** to configure the observed URLs resp. 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.

Select to **Display revisions for the last** entered **days**. You can further limit the number of displayed revisions by **But at**.

- Choose **Least** to have at least the specified number of **Revisions** reported. If there have been less revisions within the last **days** the display period will be extended so that at least the specified number of revisions are displayed. If there have been more revisions within the last **days**, this option won't affect the display.
- Choose **Most** to have at most the specified number of **Revisions** reported. If there have been less revisions within the last **days**, this option won't affect the display. If there have been more revisions within the last **days** the display period will be shrunk so that at most the specified number of revisions are displayed.

Note	For large resp. quite active projects, using a large value for Display period without a reasonable Most restriction can require significant memory usage and computational efforts.
-------------	---

The watched URLs can be refreshed manually by **Transaction|Refresh** and they will be refreshed recurrently for the interval specified in the Preferences (see 9.11).

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 email clients: Newly fetched revisions are considered as *Unread* and hence are displayed with a blue color. In addition they will have a different icon, for details refer to Table 5.1. Use **View|Mark as Read** or **View|Mark All as Read** to mark revisions as *read*.




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

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**. Choose whether to show **Time**, **Author**, **File count** and/or **Trunk/Branch/Tag**. Choose whether to have the layout **Compact** or **Aligned in columns**.

5.1.5 Transaction menu

- Use **Refresh** to refresh the log information for the Watched URLs (see 5.1.2).
- Use **Close** to close the frame.

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 resp. view 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 2.4.2) 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 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.
- **Configure Watched URLs**, see Section 5.1.2.

5.1.7 View menu

- **Mark as Read**, see Section 5.1.3.
- **Mark All as Read**, see Section 5.1.3.
- **Ungrouped Revisions**, see Section 5.1.1.
- **Grouped by Weeks**, see Section 5.1.1.
- **Grouped by Time**, see Section 5.1.1.
- **Grouped by Author**, see Section 5.1.1.
- **Settings**, see Section 5.1.4.

5.1.8 Modify menu

- **Change Commit Message**, see Section 8.5.4.

5.1.9 Query menu

- Use **Show Changes** to display the changes for the selected file or revision. For details refer to Section 3.9.1.
- Use **Log** to display the log for the currently selected revision or file. For details refer to Section 3.9.4.
- Use **Revision Graph** to display the revision graph for the currently selected revision or file. For details refer to Section 3.9.5.
- Use **Annotate** to display an annotated view of the currently selected file's content. For details refer to Section 3.9.6.
- 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 2.4.13 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 3.9.4) transparently works on the the project files and directories as well as on Transaction revisions resp. files (Pro Only). The Transactions-specific commands can be found in the **Transactions** menu, see Section 5.2.2.

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** (Pro Only).

For working copy revisions, their read/unread (see 5.1.3) state (Pro Only) 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 correspondingly (Pro Only): 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 3.3.1) 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** (Pro Only) 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** 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.

5.2.2 Transactions Menu

- **Refresh**, see Section 5.1.5.
- **Mark as Read** (Pro Only), see Section 5.1.7.
- **Mark All as Read** (Pro Only), see Section 5.1.7.
- Select **Show Branches and Tags** (Pro Only) 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** (Pro Only) to display not only the working copy revisions but also revisions which have explicitly been configured to be watched by **Configure Watched URLs**.
- **Grouped by Weeks**, see Section 5.1.1.
- **Grouped by Time**, see Section 5.1.1.

- **Grouped by Author**, see Section 5.1.1.
- **Rollback** (Pro Only), see Section 8.5.4.
- **Change Commit Message** (Pro Only), see Section 8.5.4.
- **Configure Watched URLs** (Pro Only), see Section 5.1.2.
- **Settings**, see Section 5.2.1.

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 3.9.4) itself. It stores and supplies the raw log information as received from the server and can supply them for 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 is that logs of a certain “module” can have links to other modules, because of the way Subversion’s *Copy* mechanism works. Sometimes repositories can be very large and you are 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**.

By default, SmartSVN also caches the *mergeinfo* of the corresponding repository as it is used when performing a Log command with Include Merged Revisions (see 3.9.4) selected. For certain repositories which contain a lot of “complex” mergeinfo, it can however be very time consuming to create the Log Cache. In this case it’s better to have **Include merge info** deselected.

Tip	If you had Include merge info selected and SmartSVN does not progress with building the Log Cache, stop SmartSVN and remove the corresponding Log Cache, as explained in Section 5.3.2. After starting SmartSVN, you will be asked again whether to build a Log Cache. This time, deselect Include merge info .
------------	---

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 8.5.4).

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

5.3.1 Rebuild Log Cache

In the Project Window (see 2) use **Tools|Rebuild Log Cache** to completely rebuild a local Log Cache (see 5.3). Select the **Cache** to be rebuilt and for which **Revisions** to start the rebuild. 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. Select **Include merge info** to rebuild the cache with *mergeinfo*, for details refer to Section 5.3.

5.3.2 Storage

The Log Cache information is stored in the subdirectory `log-cache` in SmartSVN's home 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 should encounter problems when rebuilding the cache or you need to get rid of the cached information for a certain repository, you can find out corresponding subdirectory and remove it, resp. remove the whole `log-cache` if you want 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

Repository Profiles

The *Repository Profiles* contain all settings which are required to establish a connection respectively authenticate to a repository.

SmartSVN automatically creates a new profile, when opening a working copy, which contains a currently unknown repository URL. Typically, such profiles are not fully configured, because there are additional usernames, passwords or certificates required for a successful authentication. When commands are invoked, which are connecting to the repository, SmartSVN will query for this additional information.

Alternatively (and important for checkouts) the Repository Profiles can be configured from the Project Window (see 2) and from the Repository Browser (see 4) by **Repository|Manage Profiles**.

6.1 Profiles

On the **Profiles** page, you can configure the main connection settings resp. profiles. The table shows all currently known profiles. You can **Add**, **Copy**, **Edit** or **Delete** a profile.

The profiles are arranged in a specific, customizable order. This order is used for profile selectors, used within various dialogs. It also affects the search for a matching profile, when connecting to a repository; the list is searched from top to bottom. In this way you can create multiple profiles for one repository with different settings, e.g. authenticated access for certain subdirectories and anonymous access for the whole repository. To change the order of the profiles, use **Move Up** and **Move Down**. Use **Sort** to sort the profiles based on the host names; sorting the profiles will keep the order between profiles for the same host.

Use **Show Passwords** to add an additional **Password/Passphrase** column which displays the stored plain text passwords for each profile. For details on passwords refer to (see 6.4).

6.1.1 Add

By **Add** a wizard will come up, which lets you supply all necessary information to create a new profile.

Configuration

On the **Location** page you have to primarily specify which **Protocol** (protocol) shall be used to access the repository. In case of **SVN+SSH**, you can optionally specify whether to **Prepend SSH login name to host**. This option is not important for SmartSVN but may be convenient when also working with the command line.

Further mandatory parameters of a profile are **Server Name**, **Repository Path** and **Server Port**. For the **Server Port** you have the option to use the **Default** port, or use a **Non-Default** port.

Note	The Repository Path is interpreted differently depending on the Protocol . For HTTP , HTTPS it denotes the <i>Location</i> as specified in <i>Apache's httpd.conf</i> (or child configuration files). For SVN it denotes the path relative to the repository root, which <code>svnserve</code> serves; you will typically simply use <code>"/</code> here. For SVN+SSH it denotes the absolute file system path to the repository, i.e. the same path which you would supply for the <code>svnserve -r</code> parameter.
-------------	--

Instead of typing the values into the various input fields, you can also use **Enter SVN URL** and supply the complete URL for the repository.

Details

Depending on the selected **Protocol**, there are different options which have to be configured on the **Details** page. Most of them are related to *authentication*.

SVN For **SVN** connections, you have to specify the **SVN Login**. This can either be **Anonymous** or by **User Name and Password**. In the latter case you have to supply the **User Name** and **Password**. The **Password** can optionally be stored by **Save password**, see also Section 6.4.

If you are connecting to a local repository, i.e. either `localhost` or `127.0.0.1`, you can choose to **Automatically start 'svnserve'**. In this case, specify the local **Repository Directory** and the path to the **'svnserve'-Executable**. SmartSVN will then always try to start the corresponding `svnserve` process at the specified port before accessing this repository.

Note	For the autostarting of <code>svnserve</code> to work properly, it's necessary that <i>anonymous read access</i> for the corresponding repository has been configured. Before the process is started, SmartSVN checks for already running processes. Only if no running processing (serving the correct repository) has been found, SmartSVN will launch its own <code>svnserve</code> process. These processes will be shutdown automatically with the shutdown of SmartSVN itself. If SmartSVN is not shutdown gracefully, the <code>svnserve</code> processes will remain running and hence have to be shutdown manually.
-------------	--

HTTP For **HTTP** connections, you have also to specify the **SVN Login** and you can optionally choose a **Proxy** if you want to connect via a proxy server (see Section 6.2 for more details).

HTTPS For **HTTPS** connections, you have to specify the same parameters as for **HTTP** connections. Furthermore you have the option to **Use client authentication** if this is required by your SSL server. In this case choose the required **Certificate File** and enter the corresponding **Certificate Passphrase** which is used to protect your certificate. You can optionally **Save passphrase**, see also Section 6.4.

SVN+SSH For **SVN+SSH** connections, you have to specify a **Login Name** for the SSH login and you have following **Authentication** options:

- For **Password-Authentication**, enter the corresponding password. You can optionally **Save password**, see also Section 6.4.
- For **Public/Private-Key-Authentication**, enter the path to your **Private Key File** and the **Passphrase**, which is used to protect your Private Key. You can optionally **Save passphrase**.
- For **Tunnel**, select the corresponding **Tunnel**. For more details regarding *tunnels*, refer to Section 6.3.

When working over **SVN+SSH**, the username used for commit messages, etc. will default to the **Login Name**. If you prefer to use another name here, choose **Custom** for **SVN User Name** and enter the corresponding name.

Finally and common for all **Protocols** you can choose to **Verify connection when pressing 'Next'**, which is recommended.

Name

The **Name** page shows the final **URL** for the profile to be created.

For displaying on the UI, a name is assigned to every profile. Choose either **Use repository URL as profile name** or **Use following profile name** and enter a corresponding name.

Click **Finish** to create the profile.

6.1.2 Edit

When editing a profile, you can change almost all parameters which also can be entered when creating a new profile. Furthermore, on the **Advanced** page, you can specify a **Text Encoding** for the profile. This encoding will only be used by the Repository Browser commands (see 4) which are displaying file content, like **Edit|Open** or **Query|Annotate**.

6.2 Proxies

On the **Proxies** page, you can configure proxy hosts which are used to connect to SVN repositories over *HTTP/HTTPS* protocol. The configured proxies can then be used within a Repository Profile.

For a *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 **Username** and **Password**. You can choose to **Save password**, see also Section 6.4.

6.3 Tunnels

On the **Tunnels** page, you can configure custom *svn+ssh* tunnels. Tunnels are useful when already having a working SSH infrastructure which also handles authentication and communication. The configured tunnels can then be used within a Repository Profile.

A tunnel has a **Name**, a tunnel **Command** and **Parameters** for this command. The **Command** typically is an *ssh* executable, like PuTTY's `plink.exe` on Microsoft Windows or `ssh` on Unix resp. Apple Mac OS. The tunnel (resp. the command) is always invoked, when an *svn+ssh* connection is set up and handles the complete SSH-communication between SmartSVN and the server. The **Parameters** can contain predefined variables which are expanded by concrete values from the corresponding Repository Profile on the tunnel invocation:

- **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 occur in the **Parameters** definition.

6.4 Passwords

All passwords which are required to access repositories, etc. can optionally be stored in a special password store. This password store is located in the `password` file, which can be found in SmartSVN's home directory (by default the `.smartsvn` subdirectory within your main home directory).

The password store is protected by a **Master Password**, which has to be defined for the very first access of the password store. For subsequent accesses of the password store, it has to be entered to retrieve the stored passwords. You may choose to **Don't use a master password**, if you don't want to have the password store protected. However, this option is only recommended if you can make sure that the master password file itself is well protected against unauthorized access.

The master password can be changed by **Repository|Change Master Password** from within the Project Window (see 2.4.10). Use either **Change master password** to *change* the current password; this will preserve the stored passwords, but requires that you can supply the **Current Master Password**. Note that you won't need to enter the **Current Master Password**, if you are working without a master password currently. Alternatively use **Set new master password** what will remove all currently stored passwords. 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. like having **Don't use a master password** selected when initially asked to set the master password.

Chapter 7

Projects

SmartSVN internally manages your SVN working copies by “SmartSVN projects”. A SmartSVN project points to a local SVN-controlled directory and has a name and settings (Section 7.2) 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 **Create Module** (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). To create a project from an already versioned local directory, use **Create from Directory** and specify the local SVN-controlled directory.

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 by **Open** or closed by **Close**.

7.1 Project Manager

With the Project Manager (**Project|Project Manager**) 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. The project tree is displayed for the **Project|Open** dialog and the directory tree’s pop-up in the Project Window.

You can **Add** a new project. This button has the same effect as **Project|Create from Directory**. Select the local SVN-controlled root directory of the working copy for which you want to add a project and specify the corresponding **Project Name**. It’s recommended to also choose **Verify repository connection** to make sure that the corresponding repository is still valid resp. can be accessed.

Note	For a specific directory there can only exist one SmartSVN project.
-------------	---

With **Edit** you can change the **Name** and the **Root Path** of an already managed project. Choose **Reset** to reset the settings of the selected projects to the default settings

(see 7.2.4). Use **Delete** to remove projects from project tree; neither the local directory itself nor any other filesystem content will be affected by this operation.

You can rearrange the project tree directly by Drag-and-Drop which is the most convenient method. Alternatively use **Move Up** and **Move Down** to move single nodes in the hierarchy. 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. When you **Delete** a group, only this group will be deleted, but not contained projects nor groups.

Use **Add new projects** to specify where new projects should be added to the tree. This option is also used by all other commands which are adding new projects.

7.2 Project Settings

The project settings affect the behaviour of various SVN commands. Contrary to the global preferences (see Section 9), the project settings only apply to an individual project. You can edit the settings of the currently opened project by **Project|Settings**. In the top of the dialog, the **Root Path** of the project is displayed.

7.2.1 Text File Encoding

The text file encoding affects only the presentation of file contents, for instance when comparing a file (see Section 3.9.1) and it will only be used if for the file itself no *charset* has been specified by its MIME-Type property (see 3.7.2). 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 up-to-date.

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

7.2.2 Refresh/Scan

The **Initial Scan** settings specifies, whether SmartSVN scans the **Whole project** or the **Root directory only** when opening a project.

We recommend in general to use the **Whole project** option, because features like searching files in the table, etc. are relying 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 only on demand to avoid a high memory consumption.

7.2.3 Working Copy

The option **(Re)set to Commit-Times after manipulating local files** advises 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 3.3), 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 advises SmartSVN to use the *auto-props* from the SVN 'config' file, which is located in the **Subversion** directory beyond 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 3.6.1) or by the Merge from 2 sources (see 3.6.2) 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 8.3) to review merge changes in detail and you can finally use Mark Resolved (see 3.4.14) 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 3.7.7), 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. To be independent of the command line client, you can enter your own patterns by **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 3.4.1) with *binary* MIME-type (see 3.7.2). The wildcard symbols “*” and “?” can be used in the usual way.

EOL Style

This option specifies the EOL style default, which is used when adding a file (Section 3.4.1). For more details refer to Section 3.7.3.

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 3.7.2. **Report Error** will report an error.

EOL Style – Native

Usually text files are stored with 'native' EOL-Style (see 3.7.3) in the SVN repository. As a result after performing SVN operations on these files your platform's native line separator will be used ('Platform'). 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. This redefinition can be done here by choosing the desired **Native EOL-Style**.

Locks

Use **Set 'Needs Lock' for** to specify for which files the Needs Lock (see 3.10.5) should be set when they are added. With **No file**, the 'Needs Lock' property will be set to no file. With **Binary files** the property will only be set to files, which have been detected to have binary content. With **Every file** the property will be set to every file.

When committing (see 3.5) 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 recurrently refresh the files' lock states. Refer to Section 3.10.1 for details.

Keyword Subst.

This option specifies the Keyword Substitution default, which is used when adding a file (Section 3.4.1). For more details refer to Section 3.7.4.

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.

7.2.4 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 8

Subwindows

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

8.1 Text Editor

The Text Editor window shows the contents of a text file and allows modifications of the file. The Text Editor is typically invoked by **Edit|Open** from the Project Window (unless an external editor has been specified in the Preferences (see [9.10](#))).

8.1.1 File menu

- Use **Save** to save the file.
- Use **Close** to close the frame.

8.1.2 Edit menu

Contains well-known functions to alter the file content resp. to find a certain text within the content.

8.1.3 View menu

- **Settings**, refer to Section [9.8](#).

8.1.4 Go To menu

- Use **Go to Line** to go to the specified **Line Number**.

8.1.5 Window menu

Refer to Section [2.4.13](#) for more details.

8.2 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 by **Query|Show Changes** from the Project Window (see 2) but there are various other ways/windows to invoke a File Compare in SmartSVN.

Depending on the source of the compared files (local working copy, repository), none, only the right, or both contents 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 – if its contents are available.

Tip If the file compare refuses to compare a file because it's *binary*, check the corresponding MIME-Type (see 3.7.2) property. Regarding the used encoding, refer to Section 7.2.1.

8.2.1 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 9.8), not only complete lines, but also the content within lines is compared if they are not too different. These comparison results in *inner-line* changes. You can take over such changes from left to right or vice versa by **Apply Left** resp. **Apply Right** from the popup menu (invoked on the change itself).

Regarding the following menus, many of the available operations are working on the *active* view, i.e. the view which has the focus resp. displays the cursor.

8.2.2 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.

8.2.3 Edit menu

Contains well-known functions to alter the file content resp. to find a certain text within the content. Additionally:

- Use **Take Left Block** to take over the complete block below the cursor position from left to right. This may also remove resp. 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 resp. insert blocks in the left view.

8.2.4 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 9.8.
- **Settings**, refer to Section 9.8.

8.2.5 Go To menu

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

8.2.6 Window menu

Refer to Section 2.4.13 for more details.

8.3 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 regarded by the UI for highlighting changes and conflicts. All file contents are directly taken from the files, which SVN produces in case of conflicting changes. The Conflict Solver is invoked by **Query|Conflict Solver** from the Project Window (see 2).

The Conflict Solver works similar to the File Compare (see 8.2), see also Section 8.2.1 for details.

8.3.1 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 3.4.14).
- Use **Close** to close the frame.

8.3.2 Edit menu

Refer to Section 8.2.

8.3.3 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.
- 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 9.8.
- **Settings**, refer to Section 9.8. 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.

8.3.4 Go To menu

In the addition to the **Go To** found in the File Compare (see 8.2.1), 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.

8.3.5 Window menu

Refer to Section 2.4.13 for more details.

8.4 Change Report (Pro Only)

The Change Report is an optimized “multi-file” compare. It gives a detailed overview of changes within a set of files. A Change Report is for instance invoked by **Query|Show Changes** from the Project Window (see 2) when having a directory with multiple changed files selected. There are various other ways/windows to invoke a Change Report in SmartSVN.

The core component of the Change Report is a read-only File Compare (see 8.2) view; for details regarding the usage, refer to Section 8.2.1. The upper part of the Change Report frame consists of a directory tree and a file table, which displays the files being part of the Change Report. In the upper, right part, another table shows the individual changes for the selected files. These changes can directly be selected and viewed in the File Compare part by navigating in this table.

Depending on the type of Change Report, not all of the following actions are available from the various menus.

8.4.1 File menu

- Use **Compare** to open a File Compare (see 8.2) for the selected file. This can be useful to edit the corresponding file.
- Use **Revert** to Revert (see 3.4.13) the local changes for the selected file. This command is only available for Change Reports which contain local files.
- Use **Log** to perform a Log (see 3.9.4) for the selected file. This command is only available for Change Reports which contain local files.
- Use **Move to Change Set** to move the selected file to a Change Set (see 3.12). The **Change Set** column in the file overview table will be updated correspondingly. This command is only available for Change Reports which contain local files.
- Use **Edit Change Set Properties** to edit the properties of the Change Set (see 3.12) to which the selected file belongs. This command is only available for Change Reports which contain local files.

8.4.2 Edit menu

Refer to Section 8.2.

8.4.3 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 2.3.
- Select **Unchanged Files** to display files without changes (and corresponding parent directories) in the directory/file structure. Unchanged files may only be present for

specific kinds of Change Reports, e.g. when performing the Change Report on a local directory.

- Use **Refresh** to refresh the file contents and re-perform the comparison.
- **Ignore Whitespace for Line Comparison**, refer to Section 9.8.
- **Settings**, refer to Section 9.8.

8.4.4 Go To menu

Refer to Section 8.2.1 for details.

8.4.5 Window menu

Refer to Section 2.4.13 for more details.

8.5 Log

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

The central component of the **Log** window is the **Revisions** table, which shows the found revisions with their attributes. You can filter out certain revisions by using **Search Author and Commit Message**. To the right of the **Revisions** table, the complete **Commit Message** 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 those files and directories, which 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. E.g. the log context root for logs performed by **Query|Log** from the Project Window (see 2) is the corresponding project root directory resp. the Externals (see 3.7.6) root directory. The context root can be enlarged to the corresponding Project Root (see 3.8.1) if necessary.

Note	Because the received log information from the server does not contain information on the node kinds (file or directory) of the revisions’s modified entries, SmartSVN tries to detect them. The more log information is present, the better are the results. However, without complete log information SmartSVN may still be wrong. In this case, the entry is assumed to be a file.
-------------	--

When *merged* revisions have been requested (see Section 3.9.4), 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.

Always exactly one of the four views is “active” which is displayed by its highlighted title. Menu bar actions (as well as toolbar buttons) are always referring to the currently active view.

8.5.1 Log menu

- Use **Show More** to extend the displayed log range.
- Use **Load Properties** (Pro Only) 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 by the system properties (see [12.5](#)).
- Use **Close** to close the frame.

8.5.2 Edit menu

- Use **Open** to open the selected revision/file/directory, for details refer to [Section 2.4.2](#). This command will only be applicable for revisions of file Logs.
- Use **Copy Message** (Pro Only) to copy the commit message of the selected revision.
- Use **Copy Path** (Pro Only) to copy the path of the selected file relative to the log context root. If multiple files are selected, all paths will be copied, each on a new line.

8.5.3 View menu

- Select **Skip Unchanged Revisions** (Pro Only) 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. E.g. when creating a Tag (see [3.8](#)) of the project root, the log for every entry of that tag will contain this tag-revision.
- Select **Revision Files/Directories** to toggle the **Directories/Files** view in the lower part of the frame.
- Select **Show Only Entries Below Selected Directory** (Pro Only) to restrict the **Directories/Files** view to only those directories and files which are actually children of the logged directory.

8.5.4 Modify menu

- Use **Change Commit Message** (Pro Only) to change the commit message of the currently selected revision. Enter the new **Commit Message** and wait until SmartSVN has rebuilt the corresponding Log Cache (see 5.3), if necessary.
- Use **Rollback** (Pro Only) 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 3.5). This command will only be applicable for logs which have a link to a local working copy.

8.5.5 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 8.2) or the Change Report (see 8.4) will come up.
- Use **Compare with Working Copy** (Pro Only) to compare the selected revision/file against the file's working copy within your project. This command will only be applicable for revisions of file Logs.
- Use **Log** (Pro Only) to perform another Log for the selected file/directory. This command will not be applicable for revisions as it would result in the same log as already present.
- Use **Revision Graph** (Pro Only) to create a Revision Graph (see 8.6) for the selected revision/file/directory.
- Use **Annotate** (Pro Only) to Annotate (see 8.7) the selected revision/file. This command will only be applicable for revisions of file Logs.
- Use **Save As** (Pro Only) to save the contents of the selected revision/file to a local file, for details refer to (see 4.5). This command will only be applicable for revisions of file Logs.

8.5.6 Window menu

Refer to Section 2.4.13 for more details.

8.6 Revision Graph (Pro Only)

The **Revision Graph** window shows all entries (files/directories) within all revisions which are related to a specific repository entry (file/directory) at a specific revision. A Revision Graph is typically invoked by **Query|Revision Graph** from the Project Window (see 2), but there are various other ways/windows to invoke a Revision Graph in SmartSVN.

The central component of the **Revision Graph** window is the **Revisions** graph, which displays the complete graph for the selected entry. The graph consists of *nodes*, *branches* and *links*.

A node represents a specific entry (file/directory) at a specific revision in the repository. Every graph has a unique root node, which is displayed in the upper left corner of the graph. A node which is directly derived from another *ancestor* node, i.e. which has the same URL, but at a higher revision number, is displayed directly below its ancestor in the same *branch*. A node, which is derived from another ancestor node by *copying*, is displayed right below its ancestor in a separate branch. A node shows its revision number, author and date. It can also show *inlined* tags and branches in the lower part of the node's area. Tags and branches are copies of the revision graph entry which have happened in a specific revision, hence in general they would be represented by separate nodes on their own. They will be inlined however, if the revision graph entry itself has not been changed in the tag/branch copy revision and no further commits to the copied location have happened. To detect tags and branches, the Tag-Branch-Layout (see 3.8.1) must be configured properly.

A branch is a collection of linked nodes (which are directly derived from each other), at the same URL. The head of the branch displays this URL, divided into trunk/tag/branch, path and name of the node. The division of the URL depends on the Tag-Branch-Layout (see 3.8.1) and certain parts (like the name, or the path) may be omitted if they have not changed compared to the ancestor node.

You can navigate through the graph either with the mouse or with the keyboard (cursor keys) and select certain nodes by clicking with the mouse or using the <Space> key.

The overall layout of the window is similar to the Log (see 8.5) window. To the right of the **Revisions** component, the complete **Commit Message** of the selected node is shown. The lower **Directories/Files** shows all files/directories for the currently selected revision which are children of the **Log Scope** (see Section 3.9.5); other entries of the revision are skipped.

Always exactly one of the four views is “active” which is displayed by its highlighted title. Menu bar actions (as well as toolbar buttons) are always referring to the currently active view.

8.6.1 Graph menu

- Use **Export as Image** to export the complete **Revisions** graph to an image file.
- Use **Close** to close the frame.

8.6.2 Edit menu

Refer to Section 8.5.2 for details.

8.6.3 View menu

- Use **Zoom In** to increase the zoom level of the graph.

- Use **Zoom Out** to decrease the zoom level of the graph.
- Select **Show Dates** to toggle the display of the nodes' revision date.
- Select **Show Copy Source** to toggle the display of the nodes' copy source, if present.
- Select **Show Tags** to toggle the display of the nodes' inlined tags.
- Select **Revision Files/Directories** to toggle the **Directories/Files** view in the lower part of the frame.

8.6.4 Modify menu

Refer to Section 8.5.4 for details.

8.6.5 Query menu

Refer to Section 8.5.5 for details.

8.6.6 Window menu

Refer to Section 2.4.13 for more details.

8.7 Annotate

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

The **Revision** selector (Pro Only) displays all revisions for which the corresponding file contents are available. These will be all revisions of the file, if for the corresponding Annotate command (see 3.9.6) **Track content of all revisions** had been selected. Otherwise, only the annotated revision of the file itself will be displayed and the selector won't be applicable. So using this selector you can navigate through all contents of the file.

Change **Color By** to change the line coloring:

- 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 have the lines color based on 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 resp. 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.

8.7.1 Annotate menu

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

8.7.2 Edit menu

Contains well-known functions to alter the file content resp. to find a certain text within the content.

8.7.3 View menu

- **Settings**, refer to Section 9.8.

8.7.4 Revision menu (Pro Only)

- Use **Show File Changes** to invoke a File Compare (see 8.2) between the currently selected **Revision** and the previous revision.
- Use **Show Revision Changes** to invoke a Change Report (see 8.4) 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.

8.7.5 Go To menu

Refer to Section 8.2.5 for details.

8.7.6 Window menu

Refer to Section 2.4.13 for more details.

8.8 Merge Preview (Pro Only)

The *Merge Preview* is the result of a Merge command (see 3.6.1) 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* or *Unchanged*. For *Modified* files, both the **Content** as well as the **Properties** can be either *Conflicting*, *Modified* or *Unchanged*.

8.8.1 Merge menu

- Use **Show Changes** to show the File Compare (see 8.2) 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 8.3) 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.

8.8.2 View menu

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

8.8.3 Window menu

Refer to Section 2.4.13 for more details.

Chapter 9

Preferences

The application preferences define the global behaviour of SmartSVN, regarding UI, SVN commands, etc. Contrary to the project settings (see Section 7.2), these preferences apply to all projects.

Tip	The preferences are stored in the <code>settings.xml</code> file in SmartSVN's home directory.
------------	--

9.1 On Start-Up

These settings configure the startup behaviour 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 that the root paths of certain projects is not valid anymore, you will be asked whether to remove these projects from the project tree (see Section 7.1).

9.2 Project

For **Open Project** you can specify the behavior when opening a project: Projects can always be opened **In current window**, unless there are SVN operations active for the currently opened project, or they can always be opened **In new window**. Optionally, you may choose to **Ask when using the project popup-menu** where to open the project.

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

9.3 User Interface

These settings configure certain aspects of the user interface of SmartSVN.

Select whether to use **Basic** or **Advanced** recursion options, for details refer to Section 3.14.1.

Select **Use View-menu file filters also for directories** to have the filters from the **View-menu** within the Project Window (see 2) not only applied to files but also to directories. For details on refer to Section 2.4.3.

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

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 10.6.

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

9.3.1 Accelerators

Use this page to customize the accelerators (shortcuts) of the **Project** main window and certain subwindows.

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 .
------------	---

9.3.2 Context Menus

Use this page to customize the context menus of the Project Window.

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.

9.4 Commit

Here you can configure global commit (see 3.5) options. Choose **Remind me to enter a commit message** to make SmartSVN warn you when trying to commit without a message. Choose **Trim whitespaces from commit message** to trim leading and trailing whitespaces from the commit message directly before committing. Choose **Warn for case-changed files** to make SmartSVN warn you before trying to commit case-changed files; even when warned, you will still be able to continue with the commit.

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

Choose for **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** to receive a warning if you have selected *all visible committable* files and there are modified directories (containing properties changes).
- Select **Warn for potentially missed directories and files** to receive a warning if you have selected *all visible committable* files and there are modified directories or committable files.

9.5 Conflict Solver

Here you can configure external tools which should be used instead of the built-in Conflict Solver (see 8.3).

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

Arguments are passed to the **Command** as it would occur from the OS command line. The place holder `${leftFile}`, `${rightFile}`, `${mergedFile}` and `${baseFile}` can be used, which will be substituted by the absolute file path of the left/right resp. merged (resulting) file. Furthermore, the place holder `${encoding}` can be used which will be substituted by the file's used encoding. Refer to Section 7.2.1 for details.

9.6 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 the set of files is opened at once. It is recommended to set this value not too high, because accidentally opening a large amount of files can overextend the system.

9.7 Refresh

These settings configure the behaviour of refreshing 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.

Choose **Perform 'cleanup' if necessary** to automatically cleanup after a manual Refresh. See Section 3.4.16 for details.

By **Manual Refresh** you can configure how the manual Refresh by **View|Refresh** (see Section 2.3) behaves. All options take into account the scanned/unscanned state of the working copy, see Section 7.2.2.

- You have the option to refresh **Always root directory**. In this case the directory selection in the tree does not matter, but always the whole project is refreshed. This option requires the most effort, but will guarantee that after changing the selection in the tree, displayed data is still 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, that you are only working a specific part of your whole 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 be always aware of which directories you have refreshed and hence which information displayed in directory tree and file table are actually up to date.

SmartSVN can also automatically perform a refresh of the project after it gets the focus back, if configured by **Refresh on frame activation**. The automatic refresh behaves the same way as configured for the manual Refresh. It can be useful if you are working some time on your project (e.g. in an IDE), then decide to check and commit your changes and hence get back to SmartSVN.

You have either the option to disable automatic refresh by **Never**, have an immediate refresh by **Immediately** or have only a refresh, if SmartSVN has been inactive for at least 5 seconds by **After more than 5 seconds of deactivation**. This option is useful, if you typically switch to other applications for a short period of time and do not want to trigger automatic refresh. This last option is only available on non-Microsoft Windows platforms, as on Microsoft Windows a special native module is used, which makes the refresh more efficient and will only refresh if necessary.

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 3.11.

9.8 Built-in Text Editors

These settings are used as a default for all text-displaying and editing views of SmartSVN, like the Text Editor (see 8.1), the File Compare (see 8.2), the Conflict Solver (see 8.3), the Change Report (see 8.4) and the Annotate (see 8.7).

For the **Font** page, choose the **Font Family** and the **Font Size** to be used by SmartSVN's text components. Optionally you may choose to have a **Smooth text display**, also known as "antialiasing".

For the **Colors** page choose the various colors, used by SmartSVN's text components. You can use **Reset to Defaults** to restore the “factory defaults” for this page.

For the **Editor** page you can configure various aspects of the text editing functions.

9.8.1 View Defaults

Here you can configure basic options related to the display of the file content for *all* text components:

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.

Further options are used by the File Compare, Conflict Solver and Change Report:

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.

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`.

9.9 File Comparators

Here you can configure external file compare tools which can be used instead of the built-in File Compare (see [8.2](#)).

You can link a specific **File Pattern** to a file comparator. You can either choose to use the **Built-in text file comparator** or an **External comparator**. An external comparator is defined by the operating system **Command** to be executed, and its **Arguments**.

Arguments are passed to the **Command** as it would occur from the OS command line. The place holders `${leftFile}` and `${rightFile}` can be used which will be substituted by the absolute file path of the left resp. right file to compare. 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 parameter.

Furthermore the place holders `${leftEncoding}` and `${rightEncoding}` can be used which will be substituted by the encoding of the left resp. the right file. Refer to [Section 7.2.1](#) for details.

9.10 External Tools

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

You can link a specific **File Pattern** to an external tool. A tool is defined by the operating system **Command** to be executed, and its **Arguments**. **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. You can also choose to run the command in **SmartSVN's working directory** or in the **File's directory**.

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**.

9.10.1 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**.

9.11 Transactions

These settings configure global Transactions (see 5) settings.

For **Refresh Each** select the interval in **minutes** for which all active Transactions views shall 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**.

Refer to the system properties (see 12.4) for further configuration options which are seldom used.

9.12 Spell Checker

These settings configure the spell check support which is used primarily for the Commit command (see 3.5).

You can define multiple *Dictionaries*. Every dictionary has a **Name** which is used in the spell checker popup menu and consists of a couple of **Official Files** and optionally a personal file **File for My Own Words** which can be extended by SmartSVN.

Note	The Official Files have to be in <i>MySpell</i> format, however <i>Hunspell</i> files are in general working well too. The File for My Own Words is a simple list of words.
-------------	---

If you are using multiple **Official Files** you can choose whether to use **Only best matching** of these files or **All at the same time**. **All at the same time** is useful to combine multiple dictionary files of the same language, e.g. one file with general expressions and one with domain-specific expressions. **Only best matching** is useful to build a common dictionary containing multiple languages and have SmartSVN detect which dictionary fits better for a given text to check.

Example

When you are frequently writing *English* as well as *German* commit messages, you can specify one English and one German file for the **Official Files** and select **Only best matching**.

Now, when writing an English commit message, SmartSVN will detect after a few words that the English dictionary file fits better and hence will check the complete commit message only with the English dictionary file (as if you had manually restricted the dictionary to contain only the English file).

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

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

9.13 Shell Integration (Windows)

These settings configure the Shell Integration (see [10.4](#)) 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, either 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 only present for Fixed Drives because the display of the overlays requires a rather good performance for the when accessing the <i>SVN admin area</i> . When having working copies located on fast network shares, Icon Overlays should work here well, too. In case you have a mixture of fast network shares and e.g. slow VPN-tunneled shares, you may exclude the latter ones by the Paths input field.
-------------	---

9.14 Shell Integration (Mac OS)

These settings configure the Shell Integration (see 10.5) integration of SmartSVN.

Select whether to enable the shell integration by **Integrate in Finder** or not. If necessary, specify further **Paths** for which the shell integration shall be completely disabled. Use only plain paths, like `/Volumes`, but no patterns here.

9.15 Check for Update

These settings configure the *New Version Check* mechanism of SmartSVN (Section 2.4.14).

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.syntevo.com>. If necessary, you can specify to use a proxy server by **Use 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.

Chapter 10

Shell Integration

SmartSVN offers a *shell integration* to have the SVN functionality of SmartSVN also present in certain parts of *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.

10.1 Commands

From the shell's context menu, there are the most important SVN commands available for locally versioned files and directories. 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 the Project Settings (see 7.2), if for the current working copy directory a corresponding project exists. If no matching Project (see 7) can be found, SmartSVN will use the Default Settings (see 7.2.4).

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 are corresponding 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.
------------	--

10.2 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 10.1 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.








Icon	State	Details
	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 lead 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.
	Modified	File/directory is modified in its content or properties (compared to its revision in the repository resp. to its pristine copy) or in some other SVN state, see Table 2.5 and Table 2.1 for details.

Figure 10.1: Overlay Icons

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 10.1. If you prefer, you can replace these icons.
------------	--

The availability of overlay icons as well as commands can be configured in the Preferences (see 9.13).

10.3 Server Mode

To provide the *shell integration* without requiring SmartSVN actually being *open*, SmartSVN can be started with the `--server-mode` argument. This will just start up the core process and bring up the tray icon (see 10.6), if present.

10.4 Windows Shell Integration

The shell integration adds *overlay icons* to directory and file views of Windows and SVN commands to the context menu for directories and files. You will especially see them for the *Windows Explorer*, but also for other software which e.g. 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 10.3).

Uninstallation

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.

10.5 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 **Finder Integration** page of the Preferences (see 9.14).

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 relogin 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 Login Item. Note, that the **Hide** option has no effect. If SmartSVN is defined as Login Item, it will be started in server mode (see 10.3).

10.6 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.

You can disable the tray icon in the Preferences (see 9.3) 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 10.3).
-------------	---

Chapter 11

Installation and Files

SmartSVN stores its configuration files per-user. By default, these files are located in the `~/.smartsvn` directory. On Microsoft Windows, this is `%USERPROFILE%`. On Unix/Apple Mac OS this is simply `~`. Every *major* SmartSVN versions stores its settings in a corresponding subdirectory so multiple versions of SmartSVN can be run at the same time.

Tip	You can change the directory where the configuration files are stored by the system property <code>smartsvn.home</code> (see 12.1).
------------	--

Following configuration files of SmartSVN are notable:

- `accelerators.xml` stores the accelerators (see [9.3.1](#)) 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 [6.4](#)) used throughout SmartSVN.
- `project-defaults.xml` stores the default project settings (see [7.2.4](#)).
- `projects.xml` stores all configured projects (see [7](#)), including their settings.
- `repositories.xml` stores the Repository Profiles (see [6](#)), except the corresponding passwords.
- `settings.xml` stores the application-wide Preferences (see [9](#)) of SmartSVN.
- `tag-branch-layouts.xml` stores the configured Tag-Branch-Layouts (see [3.8.1](#)).
- `transactionsFrame.xml` stores the configuration of the Transactions frame (see [5.1](#)).
- `uiSettings.xml` stores the context menu (see [9.3.2](#)) configuration.

11.1 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, following files will be copied from the **default** directory to his private configuration area:

- `accelerators.xml`
- `project-defaults.xml`
- `repositories.xml`
- `settings.xml`
- `tag-branch-layouts.xml`
- `transactionsFrame.xml`
- `uiSettings.xml`

The **license** file (only for *Enterprise* 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.

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 license file into the default directory.
-------------	--

Chapter 12

System properties/VM options

Some very fundamental options, which have to be known early at startup time or which typically need not to be changed are specified by Java VM options instead of SmartSVN preferences.

Options supplied to the VM are either actual *standard* or *non-standard* options, like `-Xmx` to set the maximum memory limit, or *system properties*, typically prefixed by `-D`. This chapter is mainly about SmartSVN-specific system properties.

12.1 General properties

Following general purpose properties are supported by SmartSVN.

smartsvn.home

This property specifies the directory into which SmartSVN will put its configuration files; refer to Section 11 for details. The value of `smartsvn.home` may also contain other default Java system properties, like `user.home`. It may also contain the special `smartsvn.installation` property, which refers to the installation directory of SmartSVN.

Example

To store all settings into the subdirectory `.settings` of SmartSVN's installation directory, you can set `smartsvn.home=${smartsvn.installation}\.settings`.

12.2 SVN properties

Following properties are related to the core SVN functions.

svnkit.admindir

This property specifies the name of the directory into which Subversion's administrative files are stored. By default, this is the `.svn` directory.

Example

ASP.NET does not allow directories to start with a “.”, as “.svn” does. Therefore, to use *ASP.NET* in combination with SmartSVN, you can change the administrative directory name e.g. to `_svn` by `svnkit.admindir=_svn`

smartsvn.tcp.connect-timeout

This property specifies the *CONNECT* timeout for repository connections. By default, this timeout is set to 60 seconds.

Example

With `smartsvn.tcp.connect-timeout=10` you can set the *CONNECT* timeout to 10 seconds.

smartsvn.tcp.read-timeout

This property specifies the *READ* timeout for repository connections. By default, this timeout is set to one hour, which gives the server enough time to respond to time-expensive requests. On the other hand, if a server is not responding at all, SmartSVN may block for one hour, until it reports the problem. This may be annoying under certain circumstances and hence can be changed by this property. The timeout value is specified in seconds.

Example

With `smartsvn.tcp.read-timeout=60` you can set the *READ* timeout to 60 seconds.

smartsvn.default-connection-logging

With this property you can enable the connection logging (see 2.4.14) by default for all commands. This can be useful when searching for connection-related problems, which occur only rarely. By default, this property is not enabled.

Example

Use `smartsvn.default-connection-logging=true` to enable connection logging by default.

smartsvn.http-spool-directory

With this property you can define a “pool” directory into which HTTP connection data is temporarily spooled. Spooling is the process in which server response is completely (fully) read first and only then processed. This approach may result in a certain initial delay (notifications usually displayed for certain operation will only be displayed after all data is fetched), but may be necessary in case the server uses to close connection when data it provides is not completely read in a prompt manner.

In general spooling does not result in a slow down, as file system access is much faster compared to network access; the main drawback of spooling is that no events are generated while data is spooled, so you may perceive that the operation is significantly slower than with spooling turned off.

Example

Use `smartsvn.http-spool-directory=c:/temp/smartsvn` on Windows or `smartsvn.http-spool-directory=/tmp/smartsvn` on Unix or Mac OS to enable HTTP connection spooling.

12.3 User interface properties

Following properties are related to the user interface of SmartSVN.

smartsvn.lookAndFeel.usePlatformIndependent

This property switches to SmartSVN's own, *platform independent* Look'n'Feel.

Example

To use the own Look'n'Feel, set `smartsvn.lookAndFeel.usePlatformIndependent=true`

smartsvn.lookandfeel

This property specifies the Look'n'Feel of SmartSVN. The value must be the fully qualified class name of a valid Look'n'Feel on your system.

Example

On Microsoft Windows, you can change to the default Windows Look'n' feel by setting `smartsvn.lookandfeel=com.sun.java.swing.plaf.windows.WindowsLookAndFeel`

Note SmartSVN's Look'n'Feel has been optimized for SmartSVN. Changing the Look'n'Feel may result in the GUI less nice looking.

smartsvn.ui.font

This property specifies the font family which is used for SmartSVN's own Look'n'Feel. The value must be a valid Java font name.

Example

To change the font family to *Dialog*, you may use `smartsvn.ui.font=Dialog`

smartsvn.ui.fontsize

This property specifies the font size which is used for SmartSVN's own Look'n'Feel. The value specifies the *point* size of the font, which defaults to 12.

smartsvn.ui.brightness

This property specifies the brightness of menu bars, toolbar, dialog backgrounds, etc. Valid values are in the range of 0.0 to 1.0. This property is only applicable, if SmartSVN's own Look'n'Feel is used, i.e. `smartsvn.lookandfeel` has not been changed.

smartsvn.ui.window-background-brightness

This property specifies the brightness of the “White” of window backgrounds, like the file table. Valid values are in the range of 0.0 to 1.0. This property is only applicable, if SmartSVN’s own Look’n’Feel is used, i.e. `smartsvn.lookandfeel` has not been changed.

smartsvn.lookAndFeel.tooltipDisplayDuration

This property specifies the duration in seconds for displaying a tooltip.

12.4 Transaction-related properties

There are following VM properties related to the Transactions (see 5) views.

smartsvn.transaction.message-length

This property specifies the maximum commit message length which will be displayed for Transactions. Longer commit messages will be truncated to save memory usage. The default value is set to 256.

smartsvn.transactions.connect-timeout

This property specifies the *CONNECT* timeout for repository connections established by the Transactions. The default value is set to 10 seconds. For details refer to `smartsvn.transactions.connect-timeout` (see 12.2).

smartsvn.transactions.read-timeout

This property specifies the *READ* timeout for repository connections established by the Transactions. The default value is set to 60 seconds. For details refer to `smartsvn.transactions.read-timeout` (see 12.2).

smartsvn.logcache.refresh-chill-out-cycle

This property specifies the *chill out cycle* (in counts of revisions) for building the Log Cache (see 5.3). It can be used to alleviate the server in perspective that many clients will be building the cache at the same time. The default value is set to 0 revisions, meaning no chill out cycle.

Warning! Use this property only if necessary; it can slow down the build process of a Log Cache significantly, making it even unusable.

smartsvn.logcache.refresh-chill-out-seconds

This property specifies the maximum number of seconds to *sleep* during a *chill out cycle* for building the Log Cache (see 5.3). This property is only used in combination with `smartsvn.logcache.refresh-chill-out-cycle`. The default value is set to 10 seconds.

Example

Use `smartsvn.logcache.refresh-chill-out-cycle=1000` and `smartsvn.logcache.refresh-chill-out-seconds=60` to have SmartSVN sleeping 60 seconds after every 1000 received revisions.

12.5 Other properties

There are following other VM properties available.

smartsvn.logcache.useURLasUUID

The Log Cache (see 5.3) uses *repository UUIDs* to distinguish between different repositories resp. to detect whether two repositories are equal even when different URLs are used to access them. This for instance happens when using different protocols, like `ssh://` and `https://`.

Although not recommended, sometimes a repository has been created from another repository just by copying the raw files. In this case both repositories will have the same *UUID* what will confuse the Log Cache. For such cases the distinction between repositories has to be based on their URLs.

Example

Set `smartsvn.logcache.useURLasUUID=true` to have this property enabled.

smartsvn.log.maximum-custom-properties

This property specifies the maximum number of *custom* property columns displayed within the Log frame (see 8.5) after having invoked **Log|Load Properties**. The default value is set to 10.

12.6 Specifying VM options and system properties

Depending on your operating system, VM options resp. system properties are specified in different ways.

smartsvn.properties file

The `smartsvn.properties` file is present on all operating systems. It's located in SmartSVN's *home directory*; refer to Section 12.1 for details. All *system properties* can be specified in this file.

Note	<i>System properties</i> are VM options which would be specified by the <code>-D</code> prefix when directly providing them with the start of the <code>java</code> process. All options listed in this chapter are <i>system properties</i> and hence can be specified in the <code>smartsvn.properties</code> file.
-------------	---

Every option is specified on a new line, with its name followed by a “=” and the corresponding value.

Microsoft Windows

VM options are specified in `bin/smartsvn.vmoptions` within the installation directory of SmartSVN. You can also specify system properties by adding a new line with the property name, prefixed by `-D`, and appending `=` and the corresponding property value.

Example Add the line

```
-Dsmartsvn.http.timeout=60
```

to set the HTTP-timeout to 60 seconds.

Apple Mac OS X

System properties are specified in the `Info.plist` file. Right click the `SmartSVN.app` in the Finder and select **Show Package Contents**, double click the `Contents` directory and there you will find the `Info.plist` file. Open it in a text editor of your choice. Specify the system properties as key-string pairs in the `dict`-tag after the `key` with the `Properties` content.

Example Use the following key-string pairs

```
<key>Properties</key>
<dict>
  ...
  <key>smartsvn.http.timeout</key>
  <string>60</string>
</dict>
```

to set the HTTP-timeout to 60 seconds.

Specify a VM option by placing them in the `string`-tag to the `VMOptions` array.

Unix

System properties are specified e.g. in `bin/smartsvn.sh` within the installation directory of SmartSVN. You can specify a property by adding the property name, prefixed by `-D` and appending `=` and the corresponding property value to the `_VM_PROPERTIES` environment variable. Multiple properties are simply separated by a whitespace; make sure to use quotes when specifying several properties.

Example Add

```
_VM_PROPERTIES="$ _VM_PROPERTIES -Dsmartsvn.http.timeout=60"
```

before the `$ _JAVA_EXEC` call to set the HTTP-timeout to 60 seconds.