SmartSVN 2 Reference

SyntEvo GmbH, www.syntevo.com

2006

Contents

1 Introduction **5**

2 Main Window **6**

2.1 Projects . . . 6
2.2 Directory Tree and File Table . . . 7
2.2.1 Directory States/Directory Tree . . . 7
2.2.2 File States/File Table . . . 7
2.2.3 The Focus . . . 8
2.2.4 Refreshing . . . 8
2.2.5 Smart Open . . . 8
2.3 Menus . . . 9
2.3.1 Edit Menu . . . 9
2.3.2 Window Menu . . . 9
2.3.3 Help Menu . . . 9

3 Commands **14**

3.1 Checkout . . . 14
3.2 Create Module . . . 15
3.3 Updating . . . 16
3.3.1 Update . . . 16
3.3.2 Update More . . . 16
3.3.3 Switch . . . 16
3.3.4 Switch to URL . . . 17
3.3.5 Relocate . . . 17
3.4 Merging . . . 17
3.4.1 Merge . . . 17
3.4.2 Merge from URL . . . 18
3.5 Commit . . . 18
3.6 Local Modifications . . . 19
3.6.1 Add . . . 19
3.6.2 Ignore . . . 20
3.6.3 Remove . . . 20
3.6.4 Delete Physically . . . 20
3.6.5 Rename . . . 20
3.6.6 Move . . . 20

3.6.7 Smart Move . . . 21
3.6.8 Copy . . . 21
3.6.9 Revert . . . 21
3.6.10 Mark Resolved . . . 22
3.7 Advanced Copies . . . 22
3.7.1 Copy URL-WC . . . 22
3.7.2 Copy WC-URL . . . 23
3.7.3 Copy URL-URL . . . 23
3.8 Properties . . . 23
3.8.1 Edit Properties . . . 23
3.8.2 Change File Type . . . 24
3.8.3 Change EOL Style . . . 24
3.8.4 Change Keyword Substitution . . . 24
3.8.5 Change Executable Property . . . 24
3.8.6 Edit Externals . . . 24
3.8.7 Edit Ignore Patterns . . . 25
3.9 Tags . . . 25
3.9.1 Add Tag . . . 25
3.9.2 Add Branch . . . 25
3.9.3 Tag Browser . . . 26
3.10 Queries . . . 26
3.10.1 Compare . . . 26
3.10.2 Compare with Revision . . . 26
3.10.3 Compare 2 Files . . . 26
3.10.4 Change Report . . . 26
3.10.5 Log . . . 27
3.10.6 Annotate . . . 27
3.10.7 Create Patch . . . 28
3.10.8 Create Patch between URLs . . . 28
3.11 Locks . . . 28
3.11.1 Scan Repository . . . 28
3.11.2 Lock . . . 29
3.11.3 Unlock . . . 29
3.11.4 Show Info . . . 29
3.11.5 Change Needs Lock . . . 29
3.12 Remote State . . . 30
3.12.1 Refresh Remote State . . . 30
3.12.2 Clear Remote State . . . 30
3.13 Tools . . . 31
3.13.1 Conflict Solver . . . 31
3.13.2 Export Backup . . . 31
3.13.3 Canonicalize URLs . . . 31
3.13.4 Remove Empty Directories . . . 31
3.14 Common Features . . . 32
3.14.1 Revision input fields . . . 32

3.14.2 Repository path input fields . . . 32
3.14.3 Tag input fields . . . 32

4 Repository Browser **33**
4.1 Checkout . . . 33
4.2 Modifying the repository . . . 33
4.3 Querying the repository . . . 34

5 Repository Profiles **35**
5.1 Profiles . . . 35
5.1.1 Add . . . 35
5.2 Proxies . . . 37

6 Project Management **38**
6.1 Project Manager . . . 38
6.2 Project Settings . . . 38
6.2.1 Repository Layout . . . 38
6.2.2 Text File Encoding . . . 39
6.2.3 Refresh/Scan . . . 39
6.2.4 Working Copy . . . 39
6.2.5 TMate . . . 40
6.2.6 Default Settings . . . 40

7 Preferences **41**
7.1 On Startup . . . 41
7.2 Change Report . . . 41
7.3 Commit . . . 41
7.4 Conflict Solver . . . 41
7.4.1 Built-in . . . 42
7.5 Refresh . . . 42
7.6 File Comparators . . . 43
7.6.1 Built-in . . . 43
7.7 External Tools . . . 43
7.7.1 Directory Command . . . 44
7.8 Check for Update . . . 44

8 TMate **45**
8.1 Display . . . 45
8.2 Commands . . . 45
8.3 Log Cache . . . 46
8.4 Settings . . . 47

9 VM options **48**
9.1 General options . . . 48
9.2 SVN options . . . 48
9.3 User interface options . . . 49

9.4 Specifying options 50

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 area of application are software projects, documentation projects or website projects.

We've tried to make SmartSVN easy to use for new SVN users and powerful for advanced users. Users of SmartCVS, our successful CVS client, will find switching to SVN using SmartSVN very easy. Various convenient features will help you to make working with SVN more efficient and comfortable.

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 go to Alexander Kitaev from TMate Software (`http://www.tmate.org`), who provides the excellent base library SVNKit which SmartSVN uses for accessing Subversion repositories and working copies.

Chapter 2

Main Window

The main window is the central place when working with SmartSVN. In the center of the window, all **Directories** and **Files** of the current SVN working copy are displayed. Various SVN commands on these directories and files are provided by the menu bar and the toolbar.

In the bottom left area of the main window the **Output** window shows logged output from executed SVN commands.

In the bottom right the **TMate** window (Section 8) lists the collected revisions as they happend in the repository.

The size of the windows and their position can be arranged with the mouse. By dragging their titles, they can be undocked from one position and docked to another position. The layout of the windows is stored and recovered for future SmartSVN sessions.

At the very bottom of the main window the status bar displays various kinds of information, like information on the currently selected menu item, operation progress or the repository URL of the currently selected file/directory.

While SVN operations with a connection to the repository are processed, the progress display shows the total amount of sent and received bytes during this operation.

2.1 Projects

SmartSVN internally manages your SVN working copies by "SmartSVN projects". A SmartSVN project (subsequently only referenced as "project") points to a local SVN-controlled directory and has a name and some settings (Section 6.2) 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 main window always refers to one project. To work with multiple projects at the same time, you can open multiple main windows by clicking **Window|Open New**

Window. Already managed projects can be opened in a main window by **Open** or closed by **Close**.

2.2 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.

2.2.1 Directory States/Directory Tree

The directory tree shows the project's directories and their SVN states, which are denoted by different icons. They are listed in Table 2.1. In case of a versioned directory, the corresponding revision number is displayed after the name of the directory.

2.2.2 File States/File Table

The file table shows the project's files with their SVN states and various additional information. The meaning of the states/icons is listed in Table 2.3. 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

You can specify which file attributes shall be displayed in the file table by **View|Table Columns**, see Table 2.2. Also, the order of the table columns can be defined here, alternatively to rearranging them directly in the file table. Select **Automatically resize columns** to let SmartSVN automatically adjust the size of every column depending on its content. Select **Make this configuration the default** to have the selected configuration applied to every new project.

Tip	Certain table columns require to access additional files when scanning the file system and therefore slow down scanning. The note within the **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. Wildcard symbols '*' and '%' can be used with the usual meaning.

To search the file table for a certain file you can simply start typing the file name while the focus is in the file table. This will make a small popup come up, which displays the characters you have already entered. Again, wildcard symbols '*' and '%' can be used with the usual meaning.

State Filters

With the menu items in the **View** menu, you can also set filters to display only files which meet certain criteria. **Files From Subdirectories** enables the recursive view showing not only files from the currently selected directory but also those from subdirectories.

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

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 remain in *modified children* state. To actually ignore such files on the SVN-level you can use the Ignore command (Section 3.6.2).

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 **Remote Changed Files** selected, files will be displayed which are locally *unchanged*, but are remotely changed (see Table 3.2). This option has no effect, if **Unchanged Files** is selected.

2.2.3 The Focus

The directory tree and the file table are the central components within SmartSVN's main window. A virtual focus is always assigned to exactly one of both components. The focus and the currently selected files/directory define which commands/actions are available from the menu bar and the tool bar.

2.2.4 Refreshing

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 6.2). 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 7.5).

2.2.5 Smart Open

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

- An *unchanged*, *unversioned* or *added* file is opened with the file editor, see the **Edit**|**Open** command (Section 2.3.1) for further details.

- A *conflicting* file is opened with the Conflict Solver (Section 3.13.1).
- All other files are opened by comparing them (Section 3.10.1).

2.3 Menus

This section summarizes actions which are available from the **Edit**, the **Window** and the **Help** menu.

2.3.1 Edit Menu

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 file/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 7.7). If the file table has the focus the file 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 7.7). If no editor is configured there, the internal file editor will be launched.

Select Committable Files selects all committable files in the file table.

Note	In the Professional Version, SmartSVN allows to automatically add *unversioned* or remove *missing* files for a commit. Hence these files are also selected.

Select Directory selects the deepest common directory for all selected files in the file table.

Copy File Path copies the path of the selected file to the system clipboard. If multiple files are selected, all paths will be copied, each on a new line.

Copy File Name copies the name of the selected file to the system clipboard. If multiple files are selected, all names will be copied, each on a new line.

Preferences shows the application preferences (see Section 7.8).

2.3.2 Window Menu

With **Open New Window** you can open a new main window to work with multiple projects at the same time. The subsequent content of the **Window** menu depends on which windows (frames) are currently open. For each window, there is a menu item to switch to it.

2.3.3 Help Menu

Help Topics shows the online version of SmartSVN's help.

License Information shows information on your SmartSVN license and the licensing conditions for SmartSVN.

Register lets you upgrade SmartSVN to the professional version. You will need to purchase a license file, but it's definitely worth the money!

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.

Check for Updates 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 update checking within the Preferences (Section 7).

About SmartSVN shows information on the current SmartSVN version.

Icon	Meaning	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.)
	Modified children	At least one direct or indirect child of this directory has a Non-Unchanged state.
	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 or External	Directory is either the project root or an external root. This state might be combined with other directory states.
	Nested Root	Directory is a nested working copy root, but no external.
	Obstructed	The local working-copy is damaged here; backup your modifications and do a clean checkout.
	Remote	Directory only exists in the repository. This state is only used for the remote state command (see Section 3.12).
	Unscanned	Directory has not been scanned yet (see Section 2.2.4).
	Switched	Directory is switched (compared to its parent); see Section 3.3.3. Is combined with other states.
	Locked	Directory is locked *locally* because an operation has been interrupted before. A Refresh (Section 2.2.4) should fix the problem. Is combined with other states.

Figure 2.1: 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
Remote State	-	Remote state of the file (see Section 3.12)
Lock	Lock Owner	Lock state of the file (see Section 3.11)
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 3.8.2)
EOL	svn:eol-style	End-Of-Line Type of the file (see Section 3.8.3)
Keyw.	svn:keywords	Keyword substitution options of the file (see Section 3.8.4)
Needs Lock	svn:needs-lock	Whether the file should be locked before working (see Section 3.11.5)
Executable	svn:executable	Whether the file has the executable property set (see Section 3.8.5)
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 *Copied* state
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

Figure 2.2: File Attributes

Icon	Meaning	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, hence it 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 3.12).
	Obstructed	The local working-copy is damaged here; backup your modifications and do a clean checkout.

Figure 2.3: File States

Chapter 3

Commands

SmartSVN provides most of the command-line SVN 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 Checkout

Use the Checkout command to create a working copy from a project which is already under SVN control.

Page "Repository"

First you need to select the repository from which you want to check out a project. If you can't find the repository profile in the combobox, click the **Manage** button to add it, see Section 5 for details.

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 dive into the repository structure, for more details refer to Section 4.

Use the **Select Revision** button to define the revision you want to fetch. Of course the repository content might change when changing the revision.

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

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).

With **Check out non-recursively**, you will only checkout the previously selected repository directory, but no subdirectories. Later you may choose to checkout certain subdirectories by Update More(see 3.3.2). Non-recusive 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**.

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 this command 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 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 are able to dive into the repository by expanding the directory nodes, for more details refer to Section 4. Use the **Create Directory** to create a new directory for your project in the repository.

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 name to 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.

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 this command 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.

3.3.2 Update More

Use this command to get locally missing directories and files from the repository for a foregoing non-recursive Update or Checkout(see 3.1).

Update More checks for the currently selected directory, whether there are not yet checked out subdirectories. They are presented in a list and you can select one or more of them to update. **Recurse into subdirectories** specifies, whether the selected directories shall be updated resp. checked out recursively or not.

3.3.3 Switch

Use the Switch command to switch the selected directory to the Trunk, a tag or a branch.

Select **Trunk** to switch back from a branch or tag to the main trunk. Select **Branch** or **Tag** and enter the branch or tag name to switch to a tag or branch. Select **Recurse into subdirectories** to perform the switch command not only for the currently selected directory, but also for all subdirectories.

This Switch command only works on directories, which are covered by the **Repository Layout** (Section 6.2.1), which is defined in the Project Settings. To switch to arbitrary locations or other revisions than **HEAD**, use the Switch to URL command.

3.3.4 Switch to URL

Use the Switch-to-URL command to switch the selected directory to an arbitrary repository URL/revision.

Select the **Repository Profile** which you want to switch to and enter the desired **Repository Path**. To switch to the latest revision, select **HEAD**. To switch to another revision, select **Revision** and enter the revision number. Select **Recurse into subdirectories** to perform the Switch command not only for the currently selected directory, but also for all subdirectories.

3.3.5 Relocate

Use the Relocate command to "locally switch" the selected directory to another repository/URL.

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 Merging

Merging is used to incorporate changes between two repository revisions into the current working copy.

3.4.1 Merge

Use the Merge command to merge changes from other source-branches to the selected file/directory.

Select **Trunk** to merge from the main trunk. Select **Branch** or **Tag** and enter the branch or tag name to merge changes from a branch or tag. **Location** shows the final merge source.

You can either merge a **Revision Range**, where all revisions **From** a certain revision **To** another revision are merged. For the **To** Revision, you can either specify the **Revision** directly or merge up to **HEAD**. The merged changes correspond exactly to the difference

between both revisions. You can also merge changes from exactly one **Single Revision**, which is specified by its **Number**.

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

By default, merging takes the ancestry into account. This means, that Merge 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 usecases of Merge, this behaviour leads to the expected results. For all other cases, you can decide to switch it off by selecting **Ignore ancestry**.

Select **Recurse into subdirectories** to merge not only the files in the currently selected directory, but also those from subdirectories.

This Merge command only works on directories, which are covered by the **Repository Layout** (see Section 6.2.1), defined in the Project Settings. To merge from arbitrary locations use the Merge from URL command.

3.4.2 Merge from URL

Use the Merge from URL command to merge changes between two arbitrary URLs to the selected file/directory.

Profile 1, corresponding **Path 1** and **Revision 1** define the first merge source. **Profile 2**, corresponding **Path 2** and **Revision 2** define the second merge source. The difference between first and second merge source (the order of the sources is important) will be merged to the local working copy.

For details regarding the options, refer to Section 3.4.1.

3.5 Commit

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

Page "Configuration"

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

Select **Keep Locks** to keep existing file locks remaining after the commit. If not selected, the files will be automatically unlocked after a successful commit. For more information regarding locking, see Section 3.11.

Select **Automatically add unversioned and remove missing files** if you want SmartSVN to automatically add unversioned (most likely new) files and remove missing (most likely obsolete) files before the commit. Select **Detect moved and renamed files** if you want SmartSVN to detect files which are most likely renamed or moved. These files will not simply be added and removed, but marked as copied. For details, refer to Section 3.6.7.

Select **Remove removed parent directories** to make SmartSVN also scan parent directories of the files/directory which have been selected for the commit. If such a parent

directory is scheduled for removal, it will automatically be included for the commit. With **Also remove empty parent directories**, all resulting emtpy parent directories will also be included for the commit. To cleanup all empty directories within your project, you can use **Tools**|**Remove empty directories**, see Section 3.13.4.

Clicking **Next** might take some time, because the file system of your project needs to be scanned for committable files.

Page "Smart Move"

This page only occurs, if the option **Detect moved and renamed files** on the **Configuration** page is selected and at least one moved or renamed file pair was detected. For details, refer to Section 3.6.7.

Page "Confirmation"

At this page you will get 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.

Finally you can enter a **Commit Message**, which will be displayed in various kinds of logs. A meaningful commit message is very helpful for you and your team to track your changes. SmartSVN also remembers recently used commit messages and even lets you get a commit message from the log by choosing **Get from Log** from the action button.

By default, SmartSVN will warn you in case of an empty commit message. You can switch this warning off in the global commit preferences(see 7.3).

Click **Finish** to commit the selected files and directories.

3.6 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.

3.6.1 Add

Use this command to schedule files or directories for beeing 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 files MIME type (**Type**), which can basically be *text* or *binary*. Further property defaults can be specified in the project settings(see 6.2).

Tip	Automatic detection can be overriden by the `binaryExtensions` entry in `settings.xml`. All extensions listed here will automatically treated as binary.

3.6.2 Ignore

Use this command to mark unversioned files or directories to be ignored. This is useful for files or directories which should not be stored under SVN control. Usually `.obj` or `.class` files or even their whole containing directories should be marked as to be ignored.

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 altering the *svn:ignore* property, which can also be edited by **Properties|Edit Ignore Patterns**. Refer to Section 3.8.7 for details.

3.6.3 Remove

Use this command 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 unversioned files. Select **Force Removal** if you wish to perform the removal of such items anyway.

3.6.4 Delete Physically

Use this command 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.

3.6.5 Rename

Use this command 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.

Regarding the **Force Removal** option, refer to Section 3.6.3.

3.6.6 Move

Use this command 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.

Tip	There is also a special variation of this commands, which works on exactly two selected files, where one of the files is missing and the other one is unversioned. SmartSVN interprets this as a "belated" move and moves the missing file to the unversioned file without displaying any dialog.

3.6.7 Smart Move

Use this command to reproduce already performed moves/renamings of files. Typically, you will not perform moves/renamings within SmartSVN itself, but with system commands, by IDEs, etc. One such external move/renaming 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/renamed file). This has especially effects on the log of both files: The log of the removed file will end at the committed revision, while the log of the added file will start at the committed revision. To preserve the relation (and hence history/log), a belated move on both files has to be performed. Smart Move can detect such already performed moves based on the file content and displays the corresponding suggestions.

From Name displays the name of the missing (i.e. old) file. **To Name** displays the name of the unversioned (i.e. new) file. If the name of the file has not changed, i.e. **From Name** equals **To Name**, **To Name** is omitted. In the same manner **From Path** displays the path of the old and **To Path** displays the path of the new file. Again, **To Path** will be omitted if it is equal to **From Path**. **Similarity** denotes the calculated likelyhood for this file pair representing an actually happened move/renaming.

If you agree to a move suggestion, keep it selected. This will establish the necassary relation between missing and unversioned file as if the file had been moved directly by SmartSVN or any other SVN client. Otherwise, if you decide that a suggestion would relate two actually unrelated files, deselect it.

Click **OK** to actually apply the selected move suggestions.

3.6.8 Copy

Use this command to create a copy of a file or directory which is already under SVN control.

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

This command will preserve the history of the copied item.

3.6.9 Revert

Use this command to revert the local changes of the selected files/directories. In case of directories you have the option to **Revert files and subdirectories recursively**. If deselected, only the properties of the directory itself will be reverted.

- Added 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.
- Copied files/directories which are not added (i.e. only copied with history) will completely be removed.
- 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.
- 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.

3.6.10 Mark Resolved

Use this command to mark conflicting files/directories as resolved. You need 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 deselected, only the property conflicts of the directory itself will be resolved.

3.7 Advanced Copies

The **Modify** menu contains three advanced copy commands, which are directly interacting with the repository.

3.7.1 Copy URL-WC

With Copy URL-WC 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.

Select the **From Repository** from which you want to copy the file/directory **Copy** at the specified **Source Revision**. 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).

3.7.2 Copy WC-URL

With Copy WC-URL you can copy the local selected file/directory to the repository. This is the foundation for creating tags, although SmartSVN provides more convenient functions for this task (see Section 3.9).

The local file/directory **Copy Local** will be copied to the repository specified by **To 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 in the repository, you also have to specify a **Commit Message**.

3.7.3 Copy URL-URL

With Copy URL-URL you can perform pure repositories copies. This is for instance a convenient and fast way to create repository tags/branches.

Select the **Repository Profile** to which the copy shall occur. **Copy From** and the **Source Revision** specify the copy source. **Copy Into** specifies the directory into which the selected **Copy From** file/directory shall be copied. **With Name** specifies the new name of the copied file/directory (last component of the path). Because the copy is directly performed in the repository, you also have to specify a **Commit Message**.

3.8 Properties

Both, files and directories can have properties attached to them. There exists a set of predefined properties, which are used by SVN itself to manage the working copy. All other properties are "user-defined" properties. Following commands are related to properties and are available from the **Properties** menu.

3.8.1 Edit Properties

This command allows you 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 hightlighted 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 click **Add**. To create a copy from another property, select the original property, change the property's name and maybe its value, then click **Add**. To change a property, select it in the table, change the name or the value and click **Apply**. To delete a property, select it and click **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.

3.8.2 Change File Type

Use this command to change the SVN-type of the selected files. The file 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". **Don't change** will leave all file types as they are currently.

The file 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 3.6.1), the file type category (*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 type of the file by this command.

3.8.3 Change EOL Style

Use this command to change the EOL style (line separator) of the selected files. The EOL style is important when updating or checking out a file. The option **Platform Dependent** uses the platform's native line separators. This option is set by default to all added text files. It is the preferred option if you develop the same project on different platforms.

3.8.4 Change Keyword Substitution

Use this command 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.

3.8.5 Change Executable Property

Use this command 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, **Don't Change** to leave the executable property as it is currently set for each file or **Non-Executable** to remove the property form the selected file.

3.8.6 Edit Externals

Use this command to define or change externals.

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**|**Edit 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`.

Tip	It is safer to always set a **Revision** to external definitions. 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 for this external.

3.8.7 Edit Ignore Patterns

Use this command to add, change or delete *local* ignore patterns for a directory, which define files/directories to be ignored within the directory. By default, the ignore patterns are only applied to the selected directory. You may also choose to apply the patterns to all subdirectories as well by **Recurse into subdirectories**. In case of recursive ignore patterns, you may alternatively consider to specify global ignore patterns within the project settings(see 6.2.4).

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

3.9 Tags

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 commands (Section 3.7). SmartSVN provides special commands for this task, which are based upon the copy commands. All of theses commands require to have your repository layout properly configured in the **Project**|**Settings** (Section 6.2.1). The commands are available from the **Tag/Branch** menu.

3.9.1 Add Tag

Use this command to create a copy ("Tag") of your local working-copy in the `tags` directory of your repository. By default, SmartSVN will fail if the specified tag already exists. Select **Overwrite existing tag, if necessary** to create the tag anyway.

This command is similar to **Modify**|**Copy WC-URL** (see Section 3.7.2), but simplifies the special task of "Tagging".

3.9.2 Add Branch

Use this command to create a copy ("Branch") of your local working-copy in the `branches` directory of your repository. By default, SmartSVN will fail if the specified branch already exists. Select **Overwrite existing branch, if necessary** to create the branch anyway. **Switch to branch** will immediately switch to the new branch after creation.

This command is similar to **Modify**|**Copy WC-URL** (see Section 3.7.2), but simplifies the special task of "Branching".

3.9.3 Tag Browser

The Tag Browser displays 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.

The tag browser is built up information from the Log Cache(see 8.3). With **Refresh** you can refresh the cache and rebuilt 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 been *indirectly* derived from the currently selected branch in the tree.

With **Removed Tags/Branches** SmartSVN will 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 (**...**).

3.10 Queries

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

3.10.1 Compare

Use this command to compare a single, local file with its pristine copy. No connection to the repository is required.

3.10.2 Compare with Revision

Use this command to compare a single, local file with another revision of the file. Select either to compare the **Working Copy** or the **Pristine Copy**. Select futhermore to compare with the **HEAD** or with a specific **Revision** from the repository.

3.10.3 Compare 2 Files

Use this command to compare two local files against each other. No connection to the repository is required.

3.10.4 Change Report

Use the Change Report on the selected files/directory to get a quick overview over all their local changes at once.

3.10.5 Log

Use this command 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 **Log corresponding Trunk/Tags and Branches locations** to log all "source-branches" of the file. This options requires to have the project's repository layout properly configured (Section 6.2.1). Then it will detect all Branches/Tags, in which the file/directory is present and include them in the log-query.

Select **Display all files/directories from found revisions** to display for each revision all items, which have been changed. Otherwise SmartSVN will only display the logged file resp. the logged directory with its children (recursively). Selecting this option is useful, when you want to understand the context of a concrete change; for instance all files involved in a bug fix, etc.

On the **Advanced** page, you can configure the usage of the Log Cache(see 8.3). 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 revision 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 8.3 for details.

In addition to the time-dependent revision limitation from the **Configuration page**, you can limit the maximum number of returned revisions by **Don't show more than: revisions**. If present, this value is directly passed to the server resp. Log Cache, so it can increase log performance and reduce network traffic.

With **Skip resulting unchanged revisions**, SmartSVN will report only those revisions, where the selected file or the selected directory (including its children) has actually been changed. This is typically not the case, if the parent directory of the selected file or directory has been copied. Such revisions are reported by default, but can be skipped by enabling this option.

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.

3.10.6 Annotate

Use the Annotate command to view the "history" of the selected file on a per-line basis.

Similar to the Log command (see Section 3.10.5), you can configure the period of time for which the annotated view shall be calculated.

After performing this command, an **Annotate** window of the selected file will come up. It shows each line prefixed by the line number, revision number, author and date.

These values are corresponding to the revision for which the line has been added or has been changed for the last time. You can use the **Color By** to change the way of the line coloring. With **Revision** two colors are used, denoting lines older or younger than the selected revision. **Age** fades colors from blue to red denoting the age of the line. The age itself is either linearily interpolated by the corresponding **Revision** or by the actual **Time**. With **Author**, each line gets the color of its author, where author colors are randomly assigned.

3.10.7 Create Patch

Use the Create Patch command 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** to create the patch recursively for all files within the selected directory.

3.10.8 Create Patch between URLs

Use the Create Patch between URLs command to create a "Unidiff" patch between to arbitrary URLs. See Section 3.10.7 for more details on patches.

The base URL is specified by **Profile 1**, **Path 1** and **Revision 1**. The URL to which the differences shall be calculated is specified by **Profile 2**, **Path 2** and **Revision 2**. The patch itself will be written to the local **Output File**.

By default, Create Patch takes the ancestry into account. This means, that Create Patch 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**.

Select **Recurse into subdirectories** to patch not only the files in the directory itself which has been specified by the URL, but also those from subdirectories.

3.11 Locks

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

For each file, its lock state is displayed in the file table column **Lock**. For the list of possible lock states, refer to Table 3.1.

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 3.11.1), as the lock might actually be "Stolen" or "Broken".

3.11.1 Scan Repository

With this command, 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**.

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 3.1: Lock States

This column is automatically made visible, if necessary.

3.11.2 Lock

With the Lock command, 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).

3.11.3 Unlock

With the Unlock command, you can unlock the selected files 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.

3.11.4 Show Info

This commands shows information on the lock state (in repository) of the selected file.

State shows the current lock state (see Table 3.1). **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 3.11.5). **Comment** is the lock comment, as entered by the user at the time of locking.

3.11.5 Change Needs Lock

By this command, files can be marked/unmarked to require locking. This is useful to indicate to users, that they should lock the file before working with it. One aspect of this

indication is, that SmartSVN will set files which require locking (due to this property) to read-only when checking out, or updating.

3.12 Remote State

The remote state signals, what would happen in case of an update on HEAD (see Section 3.3.1). The remote state of files is displayed in the file table column **Remote State**. See Table 3.2 for the list of possible remote states of files and directories.

Name	Meaning
Latest	The local file is equal to the latest revision of this file in the repository. An update on this file will bring no changes.
Will be modified	For the local file there exists a newer revision in the repository. An update will bring the corresponding changes for this file.
Will be removed	The local file has been removed in the repository. An update will remove the file locally.
Will be added	This file does not exist locally currently in a versioned state. An update will add this file.
Obstructed	For the local file there is something wrong, either locally or locally in combination with the repository. For instance for the local file, the the latest repository revision might contain a directory with the same name.

Figure 3.2: 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 tree resp. table, which do no exist locally. To such directories and files the special local state "Remote" is assigned, see Table 2.3 and Table 2.1.

3.12.1 Refresh Remote State

With 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 the corresponding remote state is assigned. This command will also automatically show the **Remote State** column within the file table.

Additionally to the remote state, this command will also refresh the lock states of the selected files/directory (see Section 3.11).

3.12.2 Clear Remote State

Use this command to clear and hide the remote state. This will remove all directories and files which have the local state "Remote" (see Table 2.3 and Table 2.1) and hide the **Remote State** file table column.

3.13 Tools

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

3.13.1 Conflict Solver

The Conflict Solver is a kind of *Three-Way-Merge*, which can be invoked on *conflicting* files (see Table 2.3). The content of the current file (which contains the conflicts) is displayed in the center text area. 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 allows only to alter the content of the current file. The **Edit**, **Search** and **Goto** menus provide various commands for changing the file content resp. resolving the conflicts. By the **Layout** menu, you can change the layout of the Conflict Solver.

After you have successfully resolved the conflicts, you can choose **File**|**Save** to save your changes to the file. SmartSVN will detect, whether all conflicts have been resolved and in this case also automatically mark the file as resolved (see Section 3.6.10).

3.13.2 Export Backup

Use the Export Backup command to export a backup of the selected files/directory.

Root Directory displays the root directory, to which all file paths will be exported relatively. **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.

3.13.3 Canonicalize URLs

Use the Canonicalize URLs command 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 3.8.6). 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.

3.13.4 Remove Empty Directories

Use the Remove Empty Directories command 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.

3.14 Common Features

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

3.14.1 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 (**...**) behind 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 3.9 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 8.3), 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.

3.14.2 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 (**...**) behind 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.

3.14.3 Tag input fields

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 (**...**) behind the input field.

The Tag Browser (Section 3.9.3) will come up to let you select a branch or tag.

Chapter 4

Repository Browser

The Repository Browser allows direct scanning and manipulations of the repository. You can start the repository browser by **Repository**|**Browse**. Commands like Checkout or Create Module provide inbound repository browsers. Commands like Copy WC-URL provide editors, from which a repository browser can be launched.

The repository browser displays the repository content with a directory tree and a file table, similar to the Main Window. The repository file system is only scanned on demand. This happens when currently unscanned (gray) directories are expanded. To change the browsed repository, use **Repository**|**Open**.

From the **Repository** menu you can use **Select Revision** to change the browsed revision. This results in a complete refresh of the displayed repository content and may take a while. You can also explicitly refresh the content by **View**|**Refresh**. You can also **Manage Profiles**, for details refer to Section 5.

Some of the available operations can be cancelled by **Edit**|**Stop**.

4.1 Checkout

You can checkout the selected directory by **Repository**|**Checkout**. SmartSVN will then display a simplified Checkout wizard. For details refer to Section 3.1.

4.2 Modifying the repository

The **Modify** menu provides different ways for direct modifications of the repository.

You can use **Create Directory** to create a new directory in the currently selected directory. Enter **Directory Name** and a **Commit Message**, which is automatically formed, as long as you don't modify it manually.

With **Remove** you can remove the currently selected directory or file from the repository. Enter a corresponding **Commit Message**, which is automatically suggested depending on the selected file/directory. Of course, the file/directory is not destroyed, but only removed for the next revision.

Use **Copy** to create a copy of the selected file/directory. **Source Path** will be copied to **Destination Path** with the attached **Commit Message**. See also Section 3.7.3.

Use **Move** to move the selected file/directory. **Source Path** will be moved to **Destination Path** with the attached **Commit Message**.

4.3 Querying the repository

With **Edit**|**View** you view the selected file. SmartSVN will checkout the file to a temporary location and open it in the specified editor, see Section 2.3.1 for more details.

The **Query** menu provides commands to query for certain information from the repository.

With **Log** you can display the log for the currently selected directory or file. For details refer to Section 3.10.5.

With **Annotate** you can display an annotated view of a file's content. For details refer to Section 3.10.6.

Chapter 5

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 Main Window and from the Repository Browser by **Repository**|**Manage Profiles**.

5.1 Profiles

On the **Profiles** tab, 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. To change the order of the profiles, use **Move Up** and **Move Down**. The order of the profiles 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.

5.1.1 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 **Access Method** (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 **Access Method**. For **HTTP**, **HTTPS** it denotes the *Location* as specified in *Apache's* `httpd.conf` (or child configuration files). For **SVN** it denotes the path relative to the repository root, which `svnserve` serves; you will typically simply use "/" 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 `svnserve -r` 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 **Access Method**, there a different options which have to be configured on the **Details** page. Most of them are related to *authentication*.

For **SVN** connections, you have to specify the **SVN Login**. This can either be **Anonymous** or by **User Name/Password**. In the latter case you have to supply the **User Name** and **Password**. The **Password** can optionally be stored on disk by **Store password on disk**, but note that passwords are NOT SAFE.

For **HTTP** connections, you have also to specify the **SVN Login** and you can optionally choose **Use Proxy** if you want to connect via a proxy server (see Section 5.2 for more details).

For **HTTPS** connections, you have to specify the same parameters as for **HTTP** connections. Furthermore you have the option to **Enable Client Authentication** if this is required by your SSL server. In this case choose the required **Client Certificate File** and enter the corresponding **Client Certificate Passphrase** which is used to protect your certificate. You can optionally **Store passphrase on disk**, but note that passwords are NOT SAFE.

For **SVN+SSH** connections, you have to specify a **Login Name** for the SSH login. You have the option to either authenticate by **Password-Authentication** or by **Public/Private-Key-Authentication**. In case of **Password-Authentication**, enter the corresponding password. You can optionally **Store password on disk**, but note that passwords are NOT SAFE. In case of **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 **Store passphrase on disk**, but note that passwords are NOT SAFE!

Note	All passwords are stored in the `passwords.xml` file, which can be found in SmartSVN's home directory, which is by default the `.smartsvn` within your home directory. Passwords are encrypted in a simple way which is NOT SAFE! Therefore don't store valuable passwords on a machine, where other users can access `passwords.xml` file.

Finally and common for all **Access Methods** 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 repository profile. Choose either **Use Repository URL As Profile Name** or **Use This Profile Name** and enter a corresponding name.

Click **Finish** to create the profile.

5.2 Proxies

On the **Proxies** tab, you can configure a proxy, which can be used to connect to SVN repositories over HTTP/HTTPS protocol. Even if a proxy is configured, the actual use for connecting to certain repository also depends on the **Use Proxy** option within the corresponding profile's configuration (Section 5.1).

First, you have to decided whether to **Use this proxy for HTTP- and HTTPS-connections** or to completely deactivate the proxy.

For the proxy host, you need to enter **Server Name** and **Server Port**. For **Login**, select **Anonymous** if the proxy itself requires no authentication or **User Name/Password**. In the second case, specify the required **Username** and **Password**. You can choose to **Store password on disk**, but note that passwords are NOT SAFE (see Section 5.1)!

Chapter 6

Project Management

SmartSVN internally manages your SVN working copies by "SmartSVN projects", as basically described in Section 2.1.

6.1 Project Manager

With the Project Manager (**Project**|**Project Manager**) you can manage your existing SmartSVN projects.

You can **Add** a new project. This button has the same effect as **Project**|**Create Project 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**. With **Edit** you can change the **name** or **Root Path** of an already managed project. **Reset** resets the settings of the selected projects to the default settings(see 6.2.6). Projects can also be deleted by **Delete**; the local directory itself nor any other filesystem content will be touched by this operation.

You can rearrange the order of the project list with **Move Up** and **Move Down**. The specified order is used for the **Project**|**Open** dialog and the directory tree's pop-up in the main window.

6.2 Project Settings

The project settings affect the behaviour of various SVN commands. Contrary to the global preferences (see Section 7), 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.

6.2.1 Repository Layout

The **Repository Layout** defines the project's root URL (within the repository) and where the Trunk, branches and tags of the project are stored. **Trunk**, **Branches** and **Tags** must be specified relative to the **Project Root**. When using here values `trunk`, `branches` and `tags`, you are compatible with the SVN standard.

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**. This is similar for **Tags** and **Branches**.

The repository layout affects the presentation of URLs and various commands. Among them are the basic **Switch** and **Merge** commands from the **Modify**-menu and all commands related to tagging from the **Tag**-menu.

SmartSVN tries to automatically determine the repository layout when a project is opened for the first time. Nevertheless you should verify that the suggested layout actually matches your intended or already existing repository layout.

6.2.2 Text File Encoding

The text file encoding affects only the presentation of file contents, for instance when comparing a file (see Section 3.10.1). These settings are not relevant for the operations of SVN commands, 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 this encoding**.

6.2.3 Refresh/Scan

The **Initial Scan** setting 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 convenient to scan the file structure only on demand. This makes opening a project much faster and also requires less memory which can be crucial for really large projects.

6.2.4 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.

Global Ignores

The Global Ignores define which files/directories should in general be ignored within the current project. This is contrary to local ignores (see Section 3.8.7), 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 (separated with commas)**. The **Patterns** are file name patterns, where "*" and "?" are wildcard symbols, interpreted in the usual way.

Default EOL Style

This option specifies the EOL style default, which is used when adding a file (Section 3.6.1). For more details refer to Section 3.8.3.

Default 'Needs Lock'

This option specifies the 'Needs Lock' default, which is used when adding a file (Section 3.6.1). 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.

Default Keyword Subst.

This option specifies the Keyword Substitution default, which is used when adding a file (Section 3.6.1). For more details refer to Section 3.8.4.

6.2.5 TMate

Refer to Section 8 for further details.

6.2.6 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, except of the **Repository Layout** which always depends on the specific project.

Chapter 7

Preferences

The application preferences define the global behaviour of SmartSVN, regarding UI, SVN commands, etc. Contrary to the project settings (see Section 6.2), these preferences apply to all projects.

Tip	The preferences are stored in the `settings.xml` file in SmartSVN's home directory.

7.1 On Startup

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, if its root directory still exists. In case that the root path is not valid anymore, the project is removed from the list of known/managed projects (see Section 6.1).

7.2 Change Report

These settings configure the Change Report (Section 3.10.4). For further details refer to Section 7.6.1.

7.3 Commit

These settings configure the global commit(see 3.5) options. With **Remind me to enter a commit message**, SmartSVN will ask when trying to commit without a message. You can also specify the **Maximum number of previous commit messages to remember** here.

7.4 Conflict Solver

These settings configure the Conflict Solver (Section 3.13.1).

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 OS command line. Additionally the place holder `${leftFile}` and `${rightFile}` and `${mergedFile}` can be used, which are substituted by the absolute file path of the left/right resp. merged (resulting) file.

7.4.1 Built-in

With **Compare with Base** selected, the highlighting of the changes resp. conflicts takes the contents of the base file into account: Lines (in the left, center or right text area) 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.

For details on the other options refer to Section 7.6.1.

7.5 Refresh

These settings configure the behaviour of refreshing the file system.

By **Manual Refresh** you can configure how the manual Refresh by **View|Refresh** (see Section 2.2) behaves. All options take into account the scanned/unscanned state of the working copy, see Section 6.2.3.

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 if **View|Files From Subdirectories** is selected. 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.

Scrolling specifies whether to **Scroll to top of table** or **Try to keep same files visible**.

Frame activation

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-Windows platforms, as on windows as special native module is used, which makes the refresh more efficient and will only refresh if necessary.

7.6 File Comparators

These settings configure file comparators, which can be invoked by the corresponding actions from the **Query** menu.

You can link a specific **File Pattern** to a file comparator (file compare tool). 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 OS command line. Additionally the place holder `${leftFile}` and `${rightFile}` can be used, which are substituted by the absolute file path of the left resp. right file to compare. In cases, where an `./svn`-internal file like the *base file* is used for comparison, the content of this file is copied to a temporary location and this temporary file is passed as parameter.

7.6.1 Built-in

These settings configure the *Built-in text file comparator*.

The **Tab Size** specifies the width (number of characters) which is used to display a `TAB` character. The **Inner Line Comparison** specifies the "block size" to be used for comparing within lines. With **Show Whitespaces** whitespace characters will be displayed. With **Show Line Numbers** a line number gutter will be prepended. 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.

7.7 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 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 Windows `acrord32.exe`) for **Command** and leave `${filePath}` for **Arguments**.

7.7.1 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 need to **Enable Open Directory Command**. 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 Windows, to open the command shell for a selected directory, enter `cmd.exe` for **Command** and `/c start cmd.exe` for **Arguments**.

7.8 Check for Update

These settings configure the *Update-Check* mechanism of SmartSVN (Section 2.3.3).

Select **Automatically check for available updates** to make SmartSVN check for program updates after it has been started.

Chapter 8

TMate

The TMate feature is a simple version of the TMate plugin for IntelliJ IDEA (`http://www.jetbrains.co`
It collects information on repository revisions in the background and presents them in the lower left **TMate** window. The **TMate** menu provides various commands and customization facilities for the **TMate** window.

8.1 Display

The window lists the known revisions, starting with the youngest one. By **Layout** or the corresponding selector toolbar button you can switch to different predefined views. Depending on the selected view, the layout of the tree might differ, but revision nodes will always be displayed. Within a revision node the contained directories and files are displayed.

The display can be customized by the **TMate** settings, see Section 8.4 for further details.

8.2 Commands

On a file, you can use **Show Changes** to compare the file's content before and after the commit of the revision.

On a revision, you can use **Change Report** to create a Change Report for this revision. It displays all changed files with their contents and it highlights the differences between the contents before and after the commit of the revision.

On a revision, you can use **Rollback Revision** to locally rollback the revision. This will peform a reverse merge (Section 3.4.1) of the revision to the working copy. After you have verified the result, you may choose to commit the rollback.

Select in File Table tries to select the corresponding file which is selected in the TMate tree resp. all files of the selected revision from the tree in the main file table. Files from the TMate tree, for which no counterpart in the file table exists, are ignored.

On a revision, **Copy message** puts the selected commit message into the System clipboard.

Use **Refresh** to make TMate collect new information on revisions since the last known (and displayed) revision.

8.3 Log Cache

The Log Cache is the local data storage of the TMate plugin, which is also used by other SmartSVN commands, for instance the Log command (Section 3.10.5) itself. It stores and supplies the raw log information as received from the server and can supply them for various commands later on. This increases log performance significantly and also leads to reduced network traffic.

When the Log Cache is accessed for the first time for a certain repository, you may choose which parts of the repository should be indexed by the cache. In general it is advisable 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 the current module at**. However this may lead to incomplete logs due to the reasons stated above.

Keeping the Log Cache up-to-date is automatically handled by SmartSVN. All log-related commands always query the repository for the latest logs, before querying the Log Cache and the TMate plugin can update the cache frequently, if **Automatic Refresh** has been configured in the settings (Section 8.4).

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* (`svn propset --revprop`).

In such cases, you should use **Tools|Rebuild Log Cache** and select the corresponding **Cache** to completely rebuild the Log Cache and hence fetch the modified log data for the previously cached revisions.

Tip	The Log Cache is stored in the subdirectory `log-cache` in SmartSVN's home directory. 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 the corresponding numbered subdirectory or you can remove the whole `log-cache` directory. You should never change these files while SmartSVN is running, otherwise the results will be unpredictable.

8.4 Settings

The **TMate** settings are part of the project's settings (Section 6.2) and configure the behaviour of the TMate plugin and the Log Cache.

The basic refreshing behaviour of TMate resp. the Log Cache can either be **Manual refresh** or **Automatic refresh**. In the latter case you have to specify the refresh interval by **Refresh each minutes**.

Select **Refresh after loading a project** to automatically refresh the Log Cache after a project has been loaded. Select **Refresh after a command changed the working copy** to automatically refresh after Updates, Commits, etc.

Select **Show all project revisions** to display not only revisions for the current project's repository URL, but also for all other URLs (Tags/Branches), covered by the Repository Layout (Section 6.2.1).

Within the **Display** settings you can configure whether to show the revision's **Time**, **Author**, **File count** and/or **Trunk/Branch/Tag**. **Trunk/Branch/Tag** will only be displayed, if **Show all project revisions** from the main section has been selected.

Chapter 9

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. In addition to the default Java VM options, following options are supplied by SmartSVN.

9.1 General options

Following general purpose options are supported by SmartSVN.

smartsvn.home

This propery specifies the directory into which SmartSVN will put its configuration files. By default, this is `~/.smartsvn`, where `~` denotes your home directory. On Windows, this is `%USERPROFILE%`, on Unix/Mac this is simply `~`. 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 `.smartsvn` of SmartSVN's installation directory, you may set `smartsvn.home=${smartsvn.installation}\.smartsvn`.

9.2 SVN options

Following options are related to the core SVN functions.

svnkit.admindir

This option 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.http.timeout

This option specifies the timeout of *HTTP* 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.http.timeout=60` you can set the HTTP connection timeout to 60 seconds.

smartsvn.default-connection-logging

With this option you can enable the connection logging(see 2.3.3) by default for all commands. This can be useful when searching for connection-related problems, which occur only rarely. By default, this option is not enabled.

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

9.3 User interface options

Following options are related to the user interface of SmartSVN.

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

smartsvn.ui.font

This property specifies the font family which is used for SmartSVN's user interface. The value must be a valid Java font name.

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

smartsvn.ui.fontsize

This property specifies the font size which is used for SmartSVN's user interface. 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 default 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 default Look'n'Feel is used, i.e. `smartsvn.lookandfeel` has not been changed.

9.4 Specifying options

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

Windows

Options are specified in `bin/smartsvn.vmoptions` within the installation directory of SmartSVN. You can specify an option by adding a new line with the option name, prefixed by `-D`, and appending `=` and the corresponding option value.

Example Add the line

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

to set the HTTP-timeout to 60 seconds.

Unix

Options are specified e.g. in `bin/smartsvn.sh` within the installation directory of SmartSVN. You can specify an option by adding the option name, prefixed by `-D` and appending `=` and the corresponding option value to the execution call.

Example Replace

```
\$\_JAVA\_EXEC -classpath \$\_CP -Dsun.io.useCanonCaches=false
-Xmx48m SmartSVN
```

by

```
\$\_JAVA\_EXEC -classpath \$\_CP -Dsun.io.useCanonCaches=false
-Xmx48m -Dsmartsvn.http.timeout=60 SmartSVN
```

to set the HTTP-timeout to 60 seconds.

Mac OS

Options are specified in `Info.plist` within the installation directory of SmartSVN. You can specify an option by adding the option name, prefixed by `-D` and appending `=` to the `VMOptions` array. These lines must be enclosed by a `string`-tag.

Example Insert the line

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

to set the HTTP-timeout to 60 seconds.