

# SmartSVN 1.1 - A short introduction

Th. Singer and M. Strapetz, [www.smartsvn.com](http://www.smartsvn.com)

2005

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Project</b>	<b>5</b>
2.1	Project Manager . . . . .	5
2.2	Project Settings . . . . .	6
2.2.1	Repository Layout . . . . .	6
2.2.2	Working Copy . . . . .	6
2.2.3	Global Ignores . . . . .	6
2.2.4	Default Settings . . . . .	7
<b>3</b>	<b>Main Window</b>	<b>8</b>
3.1	Directory Tree and File Table . . . . .	8
3.1.1	Directory States/Directory Tree . . . . .	8
3.1.2	File States/File Table . . . . .	9
3.1.3	The Focus . . . . .	9
3.1.4	Refreshing . . . . .	9
3.2	Menus . . . . .	10
3.2.1	Edit Menu . . . . .	10
3.2.2	Window Menu . . . . .	10
3.2.3	Help Menu . . . . .	11
<b>4</b>	<b>SVN Commands</b>	<b>15</b>
4.1	Checkout . . . . .	15
4.2	Create Module . . . . .	16
4.3	Updating . . . . .	17
4.3.1	Update . . . . .	17
4.3.2	Switch . . . . .	18
4.3.3	Switch to URL . . . . .	18
4.3.4	Relocate . . . . .	18
4.3.5	Merge . . . . .	19
4.3.6	Merge from URL . . . . .	19
4.4	Commit . . . . .	19
4.5	Local Modifications . . . . .	20
4.5.1	Add . . . . .	20
4.5.2	Ignore . . . . .	21

---

4.5.3	Remove	21
4.5.4	Delete Physically	21
4.5.5	Rename	21
4.5.6	Move	21
4.5.7	Smart Move	22
4.5.8	Copy	22
4.5.9	Revert	22
4.5.10	Resolve	23
4.6	Repository Copies	23
4.6.1	Copy URL-URL	23
4.6.2	Copy URL-WC	23
4.6.3	Copy WC-URL	23
4.7	Tags	24
4.7.1	Add Tag	24
4.7.2	Add Branch	24
4.7.3	Tag Browser	24
4.8	Queries	24
4.8.1	Compare	25
4.8.2	Log	25
4.8.3	Annotate	25
4.8.4	Change Report	25
4.8.5	Create Patch	25
4.8.6	Create Patch between URLs	26
4.9	Properties	26
4.9.1	Edit Properties	26
4.9.2	Change File Type	27
4.9.3	Change Line Separators	27
4.9.4	Change Keyword Substitution	27
4.9.5	Change Executable Property	27
4.9.6	Edit Externals	27
4.9.7	Edit Ignore Patterns	28
4.10	Remote State	28
4.10.1	Refresh Remote State	29
4.10.2	Clear Remote State	29
4.11	Locks	29
4.11.1	Scan Repository	30
4.11.2	Lock	30
4.11.3	Unlock	30
4.11.4	Show Info	30
4.11.5	Change Needs-Lock	30
<b>5</b>	<b>Repository Browser</b>	<b>31</b>
5.1	Checkout	31
5.2	Modifying the repository	31
5.3	Querying the repository	32

<b>6</b>	<b>Repository Profiles</b>	<b>33</b>
6.1	Profiles . . . . .	33
6.2	SSL . . . . .	34
6.3	SSH . . . . .	34
6.4	Proxies . . . . .	35
<b>7</b>	<b>Preferences</b>	<b>36</b>
7.1	Refresh . . . . .	36
7.1.1	Refresh Behaviour . . . . .	36
7.1.2	Refresh on frame activation . . . . .	37
7.2	External Tools . . . . .	37
7.2.1	Directory Command . . . . .	37
<b>8</b>	<b>TMate</b>	<b>38</b>

# 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 this structure. Typically usecases 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 giving feature suggestions.

Special thanks goes to Alexander Kitaev from TMate Software (<http://www.tmatesoft.com>), who provides the excellent base library JavaSVN which SmartSVN relies on.

# Chapter 2

## Project

SmartSVN internally manages your SVN working copies by “SmartSVN projects”. A SmartSVN project (subsequently only denoted by “project”) points to a local SVN-controlled directory and has a name and some settings 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 4.2). 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 lives in a repository, use **Check Out** (see Section 4.1). 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 shows 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**.

### 2.1 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 function as **Project|Create Project from Directory**. Select the local SVN-controlled root directory of the working copy you want to add as project and specify a corresponding **Project Name**. To change the name of an already managed project, use **Rename**. 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.

## 2.2 Project Settings

The project settings define the behaviour of 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**.

### 2.2.1 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**. 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 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 the **Project Settings** dialog for a new project is opened for the first time. Nevertheless you should verify that the suggested layout actually matches your intended or already existing repository layout.

### 2.2.2 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 4.3), this option is convenient to get the actual change time of a file and not the local update time.

### 2.2.3 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 4.9.7), 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.

## 2.2.4 Default Settings

Projects are created by various commands. For reasons of simplicity, in most of these cases, there is no configuration possibility for the corresponding project settings. Anyway, you can specify default project settings (template settings), which will be applied to newly created projects. 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 3

## Main Window

The main window is the central place when working with SmartSVN projects. In the center of the window, all directories and files below the project's root directory 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. The Output window is cleared automatically when a new project is opened. Furthermore you have the choice to clear it manually via the clear toolbar button at any time during your work with one project.

In the bottom right the **TMate** window lists the collected revisions as they happen in the repository for the current project.

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.

### 3.1 Directory Tree and File Table

The directory tree and the file table show all local directories/files below the project's root directory. The only exception are `.svn`-directories and *ignored* directories and files within other ignored directories. Such directories/files will never be shown.

#### 3.1.1 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 3.1. In case of a versioned directory, the corresponding revision number is displayed behind the name of the directory.

### 3.1.2 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 3.3. 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 3.2. Also, the order of the table columns can be defined here, alternatively to rearranging them directly in the file table. Select **Make this configuration the default** to have the selected configuration applied to every new project.

<b>Tip</b>	Certain table columns require additional time 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.
------------	--

#### Filters

With the menu items in the **View** menu, you can also set certain filters on which files should be displayed. **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. 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.

### 3.1.3 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 directory/files define which commands/actions are available from the menu bar and the tool bar.

### 3.1.4 Refreshing

The contents of the directory tree and the file table are read into memory when a project is opened. When changes to files or directories occur from within SmartSVN, they are

refreshed automatically. In case of external changes, an explicit refresh via **View|Refresh** or the corresponding toolbar button is required. You can configure the kind of refresh (“depth”) within the application preferences (Section 7.1).

## 3.2 Menus

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

### 3.2.1 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 File** opens the selected file/directory. If the directory tree has the focus, this action is only applicable, if a **Directory Command** has been configured in the preferences (see Section 7.2). If the file table has the focus and the file is not modified or unversioned (see Table 3.3), the file will be opened in an editor (see **Edit File** command for details.) For all other file states, the internal file compare will be launched (see Section 4.8.1).

**Edit File** opens a file editor for the selected file. The editor to open can be configured in the **Externals Tools** section of the Preferences (see Section 7.2). If no editor is configured there, the internal file editor will be launched.

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

<b>Note</b>	In the Professional Version, SmartSVN allows to automatically add <i>unversioned</i> or remove <i>missing</i> files for a commit. Hence these files are also selected.
-------------	--

**Copy File Path** copies the path of the selected file to the system clipboard.

**Copy File Name** copies the name of the selected file to the system clipboard.

**Preferences** shows the application preferences (see Section 7).

### 3.2.2 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.

### 3.2.3 Help Menu

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

**License Information** shows information on your currently used license for running SmartSVN 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.

**About SmartSVN** shows some information on the current SmartSVN version.

















Icon	Meaning	Details
	Unchanged	Directory is under version control, not modified and equal to its revision in repository.
	Root or External	Directory is either the project root or an external root. This state might be combined with other directory states.
	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, i.e. differs from the repository directory for the current revision.
	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.
	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.
	Nested Root	Directory is a nested working copy root, but no external.
	Remote	This directory only exists in the repository. This state is only used for the remote state command (see Section 4.10).
	Obstructed	The local working-copy is damaged here; backup your modifications and do a clean checkout.

Figure 3.1: 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 4.10)
Lock	Lock Owner	Lock state of the file (see Section 4.11)
Last Changed Rev.	(same)	Revision, where this file has been committed
Last Changed Date	(same)	Time of the last commit of the file
Text Last Updated	(same)	Time of the last (local) update of the file's text
Properties Last Updated	(same)	Time of the last (local) update of the file's properties
Last Changed Author	(same)	Last author, i.e. who performed the last commit on the file
EOL	svn:eol-style	End-Of-Line Type of the file (see Section 4.9.3)
Executable	svn:executable	Whether the file has the executable property set (see Section 4.9.5)
Keyw.	svn:keywords	Keyword substitution options of the file (see Section 4.9.4)
Needs Lock	svn:needs-lock	Whether the file should be locked before working (see Section 4.11.5)
Type	svn:mime-type	The file's type (see Section 4.9.2)
Copy From	Copy From URL/Rev	Location and URL from which this file has been copied (locally). This value is only present if the file is in <i>Copied</i> state
Ext.	-	The file's extension
Relative directory	-	Parent directory of the file relative to the selected directory

Figure 3.2: File Attributes














Icon	Meaning	Details
	Unchanged	File is under version control, not modified and equal to its revision in repository.
	Unversioned	File is not under version control, hence it only exists locally.
	Ignored	File is not under version control (exists only locally) and marked to be ignored.
	Modified	File is modified in its content or properties, i.e. differs from the repository file for the current revision.
	Missing	File is under version control, but does not exist locally.
	Added	File is scheduled for addition.
	Removed	File is scheduled for removal.
	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. Do an update again.
	Remote	This file does not exist locally, but only in the repository. This state is only used for the remote state (see <a href="#">Section 4.10</a> ).
	Obstructed	The local working-copy is damaged here; backup your modifications and do a clean checkout.

Figure 3.3: File States

# Chapter 4

## SVN Commands

The SVN commands provide functionality to manipulate SVN aspects of your working copy. SmartSVN provides most of the command-line SVN commands in a standalone version, but also combines them to powerful higher-level commands.

### 4.1 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 6 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 5.

Use the **Filter Revisions** button to define the revision you want to fetch. Of course the repository content might change when filtering with a 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).

Click **Next** to proceed.

### Page “Project Name”

At this page you can assign a name to the project, which will automatically be created by the checkout. If you just want to check out the project without further working with it, unselect the option **Add to list of managed projects**.

Click **Next** to proceed.

### Page “Confirmation”

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

## 4.2 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 5. 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 check out the project without further working with it, unselect 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, like **File Type**, **Line Separators** and **Keyword Substitution** (**Properties** menu). After the project has been fully configured, use **Modify|Commit** to do the final import into the repository.

## 4.3 Updating

Updating from the repository can happen in various ways, by a simple update of the working copy, by switching the working copy to another repository, or by merging changes from the repository into the local working-copy. Following commands are available from the **Modify** menu.

### 4.3.1 Update

Use the Update command to get the latest changes or a specified revision from the repository.

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.

### 4.3.2 Switch

Use the Switch command to switch the working copy from operating on 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 2.2.1), defined in the Project Settings. To switch to arbitrary locations or other revisions than **HEAD**, use the Switch to URL command.

### 4.3.3 Switch to URL

Use the Switch-to-URL command to update your working copy to an arbitrary repository URL/revision.

Select the **Repository Profile** to define the repository you want to switch to. Enter the **Repository Path** or select it with the help of the Repository Browser which occurs when clicking the ellipsis button. To switch to the latest revision, select **HEAD**. To switch to a certain 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.

### 4.3.4 Relocate

Use the Relocate command to locally “switch” the working copy to another repository or root directory.

**Relocate** shows the directory, relative to the project’s root directory, which will be relocated. **From URL** shows the current full URL of this directory. For **To URL**, enter the new full URL to which the directory shall be relocated. After proceeding with **OK**, the relocation will be performed.

### 4.3.5 Merge

Use the Merge command to merge changes from other source-branches to the current working copy.

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.

Select the **From Revision** and **To Revision** to define the range of the changes to be merged. The range corresponds exactly to the difference between both revisions. If the revision number in **From Revision** is greater than the one in **To Revision**, the “reverse” changes will be merged.

By default, merging takes the ancestry into account. This can be important when you merge from one branch to another, but have renamed a file in one branch only. If you don’t need this behaviour, select **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 2.2.1), defined in the Project Settings. To merge from arbitrary or different locations or other revisions than **HEAD**, use the Merge from URL command.

### 4.3.6 Merge from URL

Use the Merge from URL command to merge changes between two arbitrary URLs to the local working copy.

**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 4.3.5.

## 4.4 Commit

Use the Commit command to write back (commit) the changes from your working copy into 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 currently locked files also locked after the commit. If not selected, the files will be automatically unlocked after a successful commit. For more information regarding locking, see Section 4.11. 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.

Enter a meaningful **Commit Message**, so you and your team mates easily can track your changes.

Clicking **Next** might take a few moments, because the file system of your project needs to be scanned.

## Page “Smart Move”

This page only occurs, if the option **Detect moved and renamed files** on the “Configuration” page is selected and a moved or renamed file pair was detected. For details, refer to Section 4.5.7.

## Page “Confirmation”

At this page you’ll get a list of all files and directories which were found to be committable. To skip a file/directory from commit, deselect the corresponding checkbox.

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

## 4.5 Local Modifications

Local commands can be performed without a connection to the repository. They are used to prepare the working copy state for a final commit. Following local commands are available from the **Modify** menu.

### 4.5.1 Add

Use this command to schedule files or directories for being added to SVN control.

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

### 4.5.2 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. This command is a shortcut for altering the `svn:ignore` property, which can also be edited by **Properties|Edit Ignore Patterns**. Refer to Section [4.9.7](#) for details.

### 4.5.3 Remove

Use this command to schedule a file or directory for being removed from SVN control.

### 4.5.4 Delete Physically

Use this command to delete local files or unversioned directories.

<b>Note</b>	Be careful before deleting a file (or directory) as there will be no way to recover unversioned items.
-------------	--

### 4.5.5 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 adding. This command will preserve the file's history.

### 4.5.6 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 adding. This command will preserve the file's history.

<b>Tip</b>	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.
------------	---

### 4.5.7 Smart Move

Use this command to reproduce an 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 detects possibly performed moves (based on the file content) and displays the corresponding files (moved **From** to **To**) as well as the likelihood, based on the **Similarity** of both files, for this move/renaming. If you agree to a move suggestion, keep it selected. This will establish the necessary 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.

### 4.5.8 Copy

Use this command to create a copy of a file or directory which is already under SVN control. This command will preserve the file's history.

### 4.5.9 Revert

Use this command to revert local changes of files or 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.

Files/directories scheduled for adding will be unscheduled. Files/directories scheduled for removal will be unscheduled. 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 as existing in the corresponding repository revision (overwriting local changes!). Missing files will be restored with their content and properties as existing in the corresponding repository revision. Conflicted files/directories will be restored with their content and properties as existing in the corresponding repository revision (ignoring local changes which caused the conflict!).

### 4.5.10 Resolve

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.

## 4.6 Repository Copies

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

### 4.6.1 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** directory shall be copied. **With Name** specifies the new name of the copied directory (first component of the path). As the copy is directly performed in the repository, you also have to specify a **Commit Message**.

### 4.6.2 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. The actual name (first component of the path) will be **With Name**.

### 4.6.3 Copy WC-URL

With Copy WC-URL you can copy your local working-copy content to the repository. This is the foundation for creating tags, although SmartSVN provides more convenient functions for this task (see Section 4.7).

The local directory **Copy Local** will be copied to the repository specified by **To Repository**. The target directory is **Into Directory**, the actual name (first component of the



path) will be **With Name**. As the copy is directly performed into the repository, you also have to specify a **Commit Message**.

## 4.7 Tags

SmartSVN simplifies the handling of “Tags” and “Branches”, which are no native SVN concepts, but managed with the help of copy commands. Following commands are available from the **Tag/Branch** menu.

### 4.7.1 Add Tag

Use this command to create a copy (“Tag”) of your local working-copy in the “tags” directory of your repository. This command is similar to **Repository|Copy WC to URL** (see Section 4.6.3), but simplifies the special task of “Tagging”. This command requires you to have your repository layout properly configured in the **Project|Settings** (see Section 2.2).

### 4.7.2 Add Branch

Use this command to create a copy (“Branch”) of your local working-copy in the “branches” directory of your repository. This command is similar to **Repository|Copy WC to URL** (see Section 4.6.3), but simplifies the special task of “Branching”. This command requires you to have your repository layout properly configured in the **Project|Settings** (see Section 2.2).

### 4.7.3 Tag Browser

Use the Tag Browser to display all tags and branches of your project at one glance. This command requires you to have your repository layout properly configured in the **Project|Settings** (see Section 2.2).

<b>Tip</b>	You can invoke the Tag Browser also from tag or branch name input fields by clicking the right ellipsis button.
------------	---

## 4.8 Queries

SmartSVN offers following non-modifying commands (locally and on the repository) by the **Query** menu.

### 4.8.1 Compare

Use this command to compare single, local files with the corresponding (backup copy of the) revision from the repository. No connection to the server is required.

### 4.8.2 Log

Use this command to display the change history of a file or directory. You can specify how far back in history the changes should be displayed.

If the Log was invoked on a directory, it will show the revisions including all affected files. If it was invoked on a file, only the revisions which contain the file will be displayed.

### 4.8.3 Annotate

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

Similar to the Log command (see Section 4.8.2), 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 linearly 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.

### 4.8.4 Change Report

Use the Change Report to get a quick overview over all your local file changes. You either can invoke it on multiple selected files or direct from a directory.

### 4.8.5 Create Patch

Use the Create Patch command to create a “Unidiff” patch for the selected files or directory. Such patches show the changes on a per-line basis of you working, 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.

### 4.8.6 Create Patch between URLs

Use the Create Patch between URLs command to create a “Unidiff” patch between to arbitrary SVN URLs. See also Section 4.8.5 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, creating patches takes the ancestry into account. This can be important when you are creating a patch from one branch to another, but have renamed a file in one branch only. If you don’t need this behaviour, select **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.

## 4.9 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.

### 4.9.1 Edit Properties

This command allows you to display and edit properties of a file or directory.

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

To enter a new property, select the last (empty) line in the table, enter the property’s name and its value, then 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**.

<b>Note</b>	Internal SVN properties starting with <b>svn</b> can’t be edited directly. However, SmartSVN offers special commands in the <b>Properties</b> menu for most of them.
-------------	--

### 4.9.2 Change File Type

Use this command to change the SVN-type of the selected files. The file type can be either a **Text** or **Binary** type, choose **Don't change** to 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** files, line endings, etc. can be replaced. By default, when adding files (see Section 4.5.1), the file type 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.

### 4.9.3 Change Line Separators

Use this command to change the line separators of the selected files. Line separators are important when updating or checking out a file. The option **Platform Dependent** uses 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.

### 4.9.4 Change Keyword Substitution

Use this command to select the keywords for the selected files, which should be substituted (expanded) locally. Keyword substitution only works for text files.

For each keyword you have the option to **Set** or **Reset** it. Select **Don't change**, to leave this special keyword set as it was.

### 4.9.5 Change Executable Property

Use this command to change the “Executable Property” of the selected files. The “Executable Property” is a versioned property, but 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(s), **Don't Change** to leave the executable property as it is currently set for each file or **Non-Executable** to remove the property from the selected file(s).

### 4.9.6 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`.

<b>Tip</b>	It is safer to always set a <b>Revision</b> to externals definitions. In this way you always know what you are currently working with and you later have the control over evaluating younger versions (revisions) of the used external.
------------	---

### 4.9.7 Edit Ignore Patterns

Use this command to add, change or delete ignore patterns for the current directory, which define files/directories to be ignored within this directory. To add an ignore pattern, you can also use the **Modify|Ignore** command.

## 4.10 Remote State

The remote state signals, what would happen in case of an update on HEAD (see Section 4.3.1). The remote state of files is displayed in the file table column **Remote State**. See Table 4.1 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 to 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 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 4.1: 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 not exist

locally. To such directories and files the special local state “Remote” is assigned, see Table 3.3 and Table 3.1.

### 4.10.1 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/directories (see Section 4.11).

### 4.10.2 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 3.3 and Table 3.1) and hide the **Remote State** file table column.

## 4.11 Locks

With 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 states is displayed in the file table column **Lock**. For the list of possible lock states, refer to Table 4.2.

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

Figure 4.2: Lock States

The “Self” state can be filled by SmartSVN when scanning the local working copy. Please note, that this state can change, when scanning the repository (see Section 4.11.1), as the lock might actually be “Stolen” or “Broken”.

### 4.11.1 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**. This column is automatically made visible, if necessary.

### 4.11.2 Lock

With the Lock command, you can lock the selected files in the repository. You can enter a **Comment**, why you have locked 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 other users, what can lead to confusion, when the other user continues working on the locked file. Hence, use this option only if necessary (e.g. if someone is on holiday and has forgotten to unlock important files).

### 4.11.3 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 was it can happen that you “break” the lock from others users, what can lead to confusion for the other one.

### 4.11.4 Show Info

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

**State** shows the current lock state (see Table 4.2). **Token ID** is the SVN Lock Token ID, which is normally to 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 4.11.5). **Comment** is the lock comment, as entered by the user at the time of locking.

### 4.11.5 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.

# Chapter 5

## 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. In contrast 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 **Filter Revisions** 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**.

### 5.1 Checkout

You can checkout the selected directory by **Repository|Checkout**. SmartSVN will then display a simplified Checkout wizard. For details refer to Section [4.1](#).

### 5.2 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 constructed, as long as you don't modify it manually.

With **Remove** you can remove the currently selected directory or file from the repository and enter a corresponding **Commit Message**. Of course, the directory/file is not



destroyed, but only removed for the next revision.

Use **Copy** to create a copy of the selected directory. The directory **Copy From** will be copied to **Destination Path** with the attached **Commit Message**. See also Section [4.6.1](#).

## 5.3 Querying the repository

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 [4.8.2](#).

With **Annotate** you can display an annotated view of a file's content. For details refer to Section [4.8.3](#).

# Chapter 6

## Repository Profiles

The Repository Profiles contain all settings which are necessary to establish a connection to a repository. They can be configured from the Main Window and from the Repository Browser by **Repository|Manage Profiles**.

### 6.1 Profiles

On the **Profiles** tab, you can configure connection settings for the SVN Repository itself. The table shows all currently known profiles. You can **Add**, **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.

For a profile you have to configure the URL by protocol type, e.g. **Http**, etc., **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. Alternatively, you can use **Enter SVN URL** to configure this part of the profile.

In cases of **Http** or **Https** connections, choose **Use Proxy** if you want to connect via a proxy server (see Section 6.4 for more details). For the **Login** you can either choose to login **Anonymously** or with **User Name/Password**. In the second case, you have the option to store the **Password** on disk.

<b>Note</b>	All passwords are stored in a file called <code>passwords.xml</code> , which can be found in SmartSVN's home directory, which is by default the <code>.smartsvn</code> 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 <code>passwords.xml</code> file.
-------------	--

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.

With **Verify connection when pressing 'OK'** selected, SmartSVN will try to connect to the specified repository. This test requires only read access. Therefore, depending on the configuration of your repository, **User Name** and **Password** might not be required and hence are not verified.

## 6.2 SSL

On the **SSL** tab, you can configure SSL connection settings, which might be required when connecting to an SVN Repository. For every host (and port) the table shows exactly one SSL configuration. You can **Edit** or **Delete** existing configurations.

In the configuration window for one host, 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 choose to **Store passphrase on disk**, but note that passwords are NOT SAFE (see Section 6.1)!

In the bottom of the window, the internally stored **MD5 Fingerprint** and **SHA Fingerprint** are displayed. Both fingerprints are calculated by the last certificate in the certificate chain (this is the certificate of the site you are connecting to), which is sent when connecting to the SSL server.

For the first connection to an SSL server, SmartSVN will ask if you are willing to trust the server by presenting these fingerprints. For subsequent connections, SmartSVN will check against those accepted fingerprints and will warn you in case of a mismatch.

## 6.3 SSH

On the **SSH** tab, you can configure SSH connection settings, which might be required when connecting to an SVN Repository. For every host (and port) the table shows exactly one SSH configuration. You can **Edit** or **Delete** existing configurations.

In the configuration window for one host, enter the **Username** for the SSH login. You have the option to either authenticate by **Password** or by a **Public Key**. In case of **Password** authentication, enter the corresponding password. You can choose to **Store password on disk**, but note that passwords are NOT SAFE (see Section 6.1)! In case of **Public Key** authentication, enter the path to your public key file and the **Passphrase**, if required to access your public key. You can choose to **Store passphrase on disk**, but note that passwords are NOT SAFE (see Section 6.1)!

## 6.4 Proxies

On the **Proxies** tab, you can configure a proxy, which can be used to connect to SVN repository over protocol HTTP/HTTPS. 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 6.1).

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 6.1)!

# Chapter 7

## Preferences

The application preferences define the global behaviour of SmartSVN, regarding UI, SVN commands, etc. Contrary to the project settings (see Section 2.2), these preferences apply to all projects.

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

### 7.1 Refresh

These settings configure the behaviour of refreshing the file system.

#### 7.1.1 Refresh Behaviour

Here you can configure how the manual Refresh by **View|Refresh** (see Section 3.1) behaves.

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.

### 7.1.2 Refresh on frame activation

SmartSVN can also automatically perform a full refresh of the project after it gets the focus back. This can be useful if you are working a 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 dll is loaded, which makes the refresh more efficient and will only refresh if necessary.

## 7.2 External Tools

These settings configure external tools, which can be invoked by the **Open/Edit** command, available from **Edit** menu.

You can link a specific **File Patterns** to externals tools. 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}` should 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 `*.txt` for **File Pattern**, the path of Acrobat Reader Executable (e.g. on Windows `acrord32.exe`) for **Command** and leave `${filePath}` for **Arguments**.

### 7.2.1 Directory Command

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

To be able to use **View|Open** on a directory, you need to **Enable Open Directory Command**. Then you can configure as for files, 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**.

# Chapter 8

## TMate

The TMate feature is a simple version of TMate's plugins (<http://www.tmatesoft.com>) for IntelliJ IDEA. It collects information on repository transactions in the background and presents them in the lower left **TMate** window.

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

The main menu bar contains a separate **TMate** menu, which allows you to **Refresh** or **Reindex** the local TMate database. **Refresh** collects new information on transactions since the last known (and displayed) transaction. **Reindex** can be used to completely rebuild the database by recollecting all information since repository birth. Reindexing is usually not necessary, but it can be helpful, if the local database should get corrupted by a system crash, etc. For a selected transaction you can open the **Change Report**, which displays the inner-file changes.

With **Settings** you can configure TMate. Select **Manual Refresh** to leave the task of updating the TMate database to yourself. Otherwise you can instruct TMate to do an **Automatic Refresh** which will be performed repeatedly after the specified **Refresh Each** time. You can also customize the display text of transactions by **Display transaction time**, **Display transaction author** and **Display transaction file count**.